НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Факультет інформатики та обчислювальної техніки
(повне найменування інституту, факультету)

кафедра обчислювальної техніки
(повна назва кафедри)

До захисту допущено
Завідувач кафедри

_____ *Г.М. Луцький*
(підпис) (ініціали, прізвище)

“___” _____ 2020 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050102 «Комп’ютерна інженерія»

на тему *«Рекомендаційна система фільмів з холодним стартом»*

Виконав: студент *4* курсу, групи *ІО-63*

Братун Андрій Юрійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *проф. Новотарський М. А.* _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант основна частина *проф. Новотарський М. А.* _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Консультант нормоконтроль *проф. Сімоненко В. П.* _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ - 2020 року

Національний технічний університет України
«Київський політехнічний інститут»

Інститут (факультет) інформатики та обчислювальної техніки
(повна назва)
Кафедра обчислювальної техніки
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 6.050102 – « Комп'ютерна інженерія»
(повна назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри

Г.М. Луцький
"___" _____ 2020 року

ЗАВДАННЯ
на дипломний проект студента

Братуна Андрія Юрійовича
(прізвище, ім'я, по батькові)

1. Тема проекту *«Рекомендаційна система фільмів з холодним стартом»*

керівник проекту *проф. Новотарський М. А.*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "_07_" _травня__ 2020 року N 1081

2. Термін подання студентом проекту _____

3. Вихідні дані до проекту технічна документація, теоретичні дані, інтернет-публікації за темою роботи

4. Зміст пояснювальної записки

- *Провести аналіїпредметної області;*
- *Провести аналіз вхідних даних та можливих гіпотез до побудови системи;*
- *Спроектувати та розробити рекомендаційну систему фільмів з холодним стартом;*
- *Провести тестування розробленої системи;*

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо)

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	*проф.Новотарський М.А.*	*16.12.2019*	*16.12.2019*
2	*проф.Новотарський М.А.*	*16.12.2019*	*16.12.2019*
3	*проф.Новотарський М.А.*	*16.12.2019*	*16.12.2019*
4	*проф.Новотарський М.А.*	*16.12.2019*	*16.12.2019*

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту	Строк виконання етапів проекту	Примітка
1	*Затвердження теми роботи*	*01.09.2019*	*виконано*
2	*Вивчення та аналіз завдання*	*02.09.2019-04.11.2019*	*виконано*
3	*Проведення аналізу предметної області*	*12.11.2019-24.01.2020*	*виконано*
4	*Проведення аналізу вхідних даних та можливих гіпотез до побудови системи*	*25.01.2020-25.02.2020*	*виконано*
5	*Проектування та розробка рекомендаційної системи фільмів з холодним стартом*	*26.02.2020-20.03.2020*	*виконано*
6	*Проведення тестування розробленої системи*	*21.03.2020-20.04.2020*	*виконано*
7	*Оформлення матеріалів роботи*	*20.04.2020-24.05.2020*	*виконано*
8	*Передзахист*	*26.05.2020*	*виконано*
9	*Захист*	*...*	*виконано*

Студент _____ Братун А.Ю.
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ Новотарський М.А.
(підпис) (прізвище та ініціали)

№ рядка	Формат	Позначення	Найменування	Кількість	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	ІАЛЦ.006305.001 ВП	Відомість дипломного проекту	1	
3	А4	ІАЛЦ.006305.002 ТЗ	Технічне завдання	3	
4	А4	ІАЛЦ.006305.003 ПЗ	Пояснювальна записка	67	
5	А3	ІАЛЦ.006305.004 Д1	Створення рекомендацій фільмів. Схема алгоритму.	1	
6	А3	ІАЛЦ.006305.005 Д2	Сховище даних. Схема даних.	1	
7	А3	ІАЛЦ.006305.006 Д3	Взаємодія моделів в системі. Схема взаємодій.	1	

Зм.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.006305.001 ВП		
Розробив		Братун А. Ю.			Рекомендаційна система фільмів з холодним стартом Відомість дипломного проекту	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Новотарський М.А.					
Н. контр.		Сімоненко В.П.				НТУУ “КПІ”, ФІОТ, ІО-63	
Затверд.							

Технічне завдання до дипломного проекту

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання розповсюджується на розробку рекомендаційної системми фільмів з холодним стартом.

Область застосування: Створення рекомендаційних листів для клієнтів будь-якого кінотеатру.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки служить покращення бізнес процесів кінотеатрів в Україні для збільшення прибутку та покращення взаємодії з клієнтами.

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є розробка рекомендаційної системи фільмів з холодним стартом.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами для розробки служать науково-технічна література з комп’ютерних технологій, публікації в періодичних виданнях, довідники з програмованих логічних інтегральних схем, публікації в Інтернеті за даним питанням.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розроблюваного продукту

- Розробка інтерфейсу для графічного відображення модуля відображення схожих фільмів.
- Створення сервісу збору та оновлення даних.
- Розробка нейронної мережі для пошуку матриці схожості фільмів
- Розробка засобів візуалізації результатів моделювання.

5.2. Вимоги до програмного забезпечення

- Операційна система MS Windows XP, MS Windows Vista, MS Windows 7, MS Windows 8/8.1, MS Windows 10
- Python 3.6.3 і вище

5.3. Вимоги до апаратного забезпечення

- Комп’ютер на базі процесору Intel Core 5 і вище
- Оперативної пам’яті не менше 16 Гбайт

Пояснювальна записка
до дипломного проекту

на тему: Рекомендаційна система фільмів з холодним стартом

Київ - 2020 року

ЗМІСТ

ВСТУП

У наш час неймовірно важкою задачею є можливість зацікавити потенційного користувача у своєму продукті для збільшення цільової аудиторії, а як наслідок, збільшення прибутку та мінімізування відтоку клієнтської бази.

Використовуючи сховище даних компанії є можливість для аналізу багатьох факторів, які можуть впливати на вибір клієнта. Але зазвичай для рішення глобальних задач потрібен глибокий аналіз даних.

Для такого аналізу використовуються зовнішні джерела даних, що може призвести до появи додаткових факторів.

Щоб опрацювати великий об’єм факторів потрібно використовувати програмне забезпечення.

На сьогодні за допомогою програмного забезпечення задачі з великим обсягом даних, які не мали рішень взагалі, вирішуються за секунди.

У нашій країні велика кількість компаній має справу з великими даними, але не використовує їх для покращення свого бізнесу. Проблема є актуальною через те, що в Україні немає належного програмного забезпечення або його ціна занадто висока.

Ця проблема існує і в кінобізнесі, оскільки немає програмного забезпечення, яке могло б у повній мірі проаналізувати вподобання користувачів для влучних рекомендацій.

По-перше, відсутнє зовнішнє джерело та API, яке б надавало якісні дані, а по-друге, кожен кінотеатр має свою цільову аудиторію, різне територіальне розташування та різний контент, що не дозволяє зробити унікальний алгоритм вирішення згаданої задачі навіть в одній мережі кінотеатрів.

Успішність кожного кінотеатру залежить від різних факторів, одним з них є вдалі маркетингові заходи, які можуть бути покращені за допомогою рекомендаційної системи, але немає жодної програми яка б могла вирішити ці задачі.

На сьогодні у світі іде сувора боротьба за можливість обробки даних, за допомогою яких можна значно спростити використання людських ресурсів а також перевіряти різні впливи факторів в тестовому оточенні.

У зв'язку з цим значно зростає потреба у створенні програмного забезпечення, яке виконувало б обробку та аналіз даних для покращення бізнес процесів.

В першому розділі описано основні методи та підходи до створення рекомендацій, розглянуто алгоритми кластеризації об'єктів для створення рекомендаційної системи з холодним стартом. Визначено структуру даних яка потрібна для використання того чи іншого алгоритму. Також розглянуто особливості поставленої задачі та чому саме ця проблема є актуальною в кінематографії.

В другому розділі було проведено дослідницький аналіз даних, де було розглянуто різні залежності в даних що могли б так чи інакше впливти на створення рекомендацій. Також було показано як саме працює частина системи яка відповідає за пошук схожих фільмів. В кінці розділу було описано основний механізм створення рекомендацій.

Третій розділ включає в себе детальний опис послідовності розробки системи, пошук джерела інформації який би надавав цілодобовий доступ до нових даних та оновлення поточних даних. Також описана розробка алгоритму пошуку матриці коефіцієнтів схожості фільмів, який обробляє різні дані. Також описано кінцевий алгоритм пост-обробки результатів за допомогою якого утворюються кінцеві рекомендації, які готові до створень рекомендаційних листів на різні види соціальних мереж.

В четвертому розділі було проведено тестування системи в три етапи. Спершу було протестовано модуль який відповідає за збір та оновлення даних. Далі було протестовано алгоритм створення матриці схожості фільмів. Останньою частиною тестування була перевірка роботи алгоритму який створює списки рекомендованих фільмів для клієнта. Також в цьому розділі показані скріншоти інтерфейсу розробленої системи.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Мета роботи: Створити програмне рішення автоматизованих рекомендацій фільмів для кожного користувача в кінотеатрі на основі статистичних даних.

Для досягнення поставленої мети були вирішені наступні завдання:

1) Дослідити предметну область створення рекомендацій користувачам, визначити недоліки існуючих підходів до вирішення даних задач

2) Виконати пошук існуючих програмних додатків автоматичного формування рекомендацій для користувачів.

3) Побудувати математичну модель системи автоматизованого формування рекомендацій, враховуючи потреби користувача

4) На основі математичної моделі створити програмний додаток для рекомендацій фільмів для користувачів.

5) Дослідити створену програмну систему автоматизованого формування рекомендацій фільмів

1.1. Методи ранжування

В чому полягає завдання ранжирування? У нас є вибірка, що складається з l елементів: x_i, ..., x_l - все це якісь об'єкти, що мають признаковий опис[1]. Якщо раніше у нас була своя відповідь для кожного об'єкта в задачах навчання з учителем, то тепер відповіді - це деякі пари з і-тих і j-тих об'єктів, для яких відомо, що і-тий об'єкт менше, ніж j-тий об'єкт. Набір таких пар і є цільовою змінною[1, 2]. Нам потрібно побудувати деяку модель a, яка приймає на вхід об'єкти, але при цьому вимоги до неї інші. Якщо раніше модель повинна була передбачити конкретну відповідь для конкретного об'єкта, то тепер модель повинна бути влаштована так, що якщо

і-тий об'єкт менше j-того, то і значення моделі на і-тому об'єкті буде менше, ніж значення моделі на j-тому об'єкті. Тобто виходи її повинні бути такими, що по ним легко відновити порядок. Нам не важливо, які саме за величиною ці відповіді, головне, щоб вони були правильно розташовані щодо один одного[3]. В цьому і полягає головна відмінність задачі ранжування від всього іншого. Найголовніший приклад застосування моделі ранжирування - це, звичайно, ранжування пошукової видачі, наприклад в Яндексі. Нам даються пари запит-документ, де запит - це деякі ключові слова, які ввів користувач в пошуковий рядок, документи - це всі документи, які є в пошуковому індексі або відібрані деяким способом. Наше завдання - навчитися відновлювати порядок на цих парах, причому порядок заданий тільки для пар, у яких один і той же запит. Тобто нам потрібно вміти ранжувати документи для одного запиту[4]. Порядок задається асессорами - це спеціальні люди, яким на вхід даються такі пари, і їх просять оцінити релевантність даного документа даному запиту, тобто те, наскільки добре даний документ відповідає на цей запит. Далі на основі цих оцінок, які можуть бути як числовими - релевантність можна оцінювати в числах, - так і попарними, тобто асесор може безпосередньо впорядкувати документи, які йому запропоновані на даний запит. Нам потрібно даний порядок вміти відновлювати, вміти відновлювати порядок документів для конкретного запиту. Ще один приклад застосування ранжирування - це завдання рекомендацій, в якій нам даються пари користувач і товар, і потрібно вміти ранжувати товари для даного користувача так, щоб вони максимізувати деяку метрику. Знову ж порядок задається тільки для одного користувача на основі того, наприклад, які переваги він має, які товари він купував, а які ні, чи оцінки які він поставив тим чи іншим товарам. Це завдання теж часто застосовується на практиці[5,6]. У завданні ранжування є деяка кількість особливостей.

По-перше, об'єкти не є незалежними - цільова змінна, відповіді залежать від пар об'єктів, і це необхідно враховувати при вирішенні. У цій

задачі набагато складніші метрики якості, ніж в ранжируванні або класифікації. Зазвичай вони навіть виявляються дискретними, оскільки їм важливі не конкретні значення моделі, а порядок, який дана модель задає.

І ще одна проблема - це те, що в даній задачі важливо правильно сформувати вибірку. Наприклад, відразу не зрозуміло, що саме віддавати на розмітку асесора, які саме пари з запитів і документів. Аналогічне питання - про рекомендації. Відразу незрозуміло, як формувати, як витягувати переваги користувачів з даних.

Всього виділяють три підходи ранжування: pointwise - поточковий та pairwise - попарний. Перший, найпростіший підхід - це поточковий. У ньому ми забуваємо про те, що цільова змінна задається на парах об'єктів, і намагаємося безпосередньо для кожного об'єкта передбачати оцінку його релевантності [6]. Наприклад, якщо ми говоримо про завдання ранжування пошукової видачі, то асесор поставив кожному об'єкту, кожній парі «запит - документ» деяку оцінку у. Її ми і будемо передбачати. При цьому ми ніяк не враховуємо, що насправді нам важлива не конкретна оцінка, а порядок, який їй задається. Проте, даний підхід простий в тому сенсі, що його можна вирішувати відомими нам методами. Наприклад, за допомогою лінійної регресії і середньоквадратичної помилки. Ми вміємо вирішувати цю задачу і отримаємо деяку модель, яка передбачає релевантність. Далі по виходах цієї моделі можна намагатися ранжувати об'єкти. Невідомо, добре це чи погано, але при цьому з якоюсь якістю завдання ми вирішимо.

Наступний підхід - попарний, або pairwise. У ньому ми все ж згадуємо, як влаштована цільова змінна, і записуємо функціонал, який залежить від пар об'єктів. Ми підсумовуємо по всім парам об'єктів таких, що $x_i < x_j$, і штрафуємо ті пари, на яких, як виявилося, що оцінка релевантності для і-го об'єкта більше, ніж оцінка релевантності для j-го об'єкта. На жаль, функціональність зменшується дискретно, і тому безпосередньо його мінімізувати не вийде [6, 7]. Але при цьому можна діяти так само, як ми діяли з класифікаторами: можна оцінити зверху даний функціонал, можна

вважати, що різниця між виходами моделі на j-м і i-му об'єкті - це відступ, і задати деяку гладку функцію від відступу L(M), і підсумовувати значення даної функції по всім парам, для яких ми знаємо порядок. Наприклад, якщо ми виберемо таку ж функцію L, як і в логістичній регресії log (1 + e (-M)), то ми отримаємо метод, який називається RankNet. Після цього можна вирішувати завдання за допомогою, наприклад, стохастичного градієнтного спуску. Так і влаштовані попарні підходи: ми записуємо функціонал, але при цьому зводимо його до вирішення простими методами, зводимо до гадкого функціоналу.

1.2. Рекомендаційні системи в сучасному світі

Рекомендаційна система - це програма яка намагається спрогнозувати які об’єкти зацікавлять користувача найбільше.

Для того щоб зробити рекомендації користувачу потрібно мати користувача, об’єкти які будуть рекомендуватися та якісь дані про взаємодію клієнта з об’єктами, наприклад оцінки користувачів які покажуть сподобався об'єкт користувачеві чи ні.

Наприклад, у завданні рекомендацій фільмів ми можемо поставити за мету спрогнозувати оцінки фільмам які користувач ще не бачив на основі оцінок які користувачі поставили фільмам, які подивилися. І на основі цих оцінок рекомендувати користувачам те, що найбільше сподобається. У цьому випадку ми маємо справу з матрицею «Користувачі-фільми» в значення якої у нас представляють оцінки, які користувачі поставили фільмам. При цьому матриця не заповнена всими числами, деякі з них пустують, бо фільм не був переглянутий [8]. Але ті, які заповнені, можуть бути заповнені як високою оцінкою, так і низькою оцінкою, що дає нам знання і про те, що людині подобається, і про те, що людині не подобається. Наше завдання в цьому випадку - заповнити комірки, в яких немає ніякої оцінки, для того щоб на

основі цих прогнозованих значень можна було рекомендувати щось користувачам.

Зовсім інша ситуація з рекомендаціями товарів. Тут ми вже не знаємо, який товар користувачеві не сподобався, тому що у нас є дані тільки про те, що користувач щось купив. У цьому випадку ми повинні будувати рекомендації, ґрунтуючись тільки на позитивному відгуці, і цей випадок називається випадком implicit feedback, а попередній випадок, коли у нас були як високі оцінки, так і низькі - випадком explicit feedback.

При цьому ми можемо вирішувати проблему побудови рекомендацій як для конкретного користувача на основі якихось його параметрів або на основі його поведінки, так і завдання побудови рекомендацій для конкретного об'єкта. Наприклад, показати схожі фільми або схожі товари, або аксесуари до цього товару. До речі кажучи, завдання рекомендацій до товару може вирішуватися за допомогою поняття взаємної інформації. Можна просто подивитися, які товари зустрічаються в призначеній для користувача сесії разом, тобто які товари користувачі дивляться разом, або які товари користувачі купують разом і на основі цього рекомендувати якісь супутні товари. Крім того, нам можуть знадобитися рекомендації в якомусь спеціальному сценарії, наприклад, часто на сайтах магазинів на головній сторінці показують найпопулярніші товари, хіти [9, 10]. При цьому, звичайно ж, поняття популярності в якійсь мірі абсолютно, тобто ми можемо сформулювати його таким чином, щоб всі товари однаково для всіх користувачів розташувати в порядку убування популярності. Але при цьому ми можемо додати якусь невелику персоналізацію, тобто трошки переставити товари з топа популярних для того, щоб показати людині саме те, що йому цікаво. А крім того, ми можемо також персоналізувати рекомендації товарів з тієї ж категорії, можемо персоналізувати блок з супутніми товарами або з товарами, які просто купують разом.

1.3. KNN та матричні підходи

Ми маємо справу з матрицею «Користувачі та об'єкти». (таблиця 1.1.) У цій матриці у нас є оцінки, поставлені об'єктам, які користувачі вже бачили. І для об'єктів, які користувачі ще не бачили, ці оцінки потрібно спрогнозувати, для того щоб рекомендувати ті фільми, які найбільш ймовірно сподобаються. Ми можемо подивитися на те, які користувачі схожі на того користувача, для якого нам потрібно зробити рекомендації, і з різними вагами, в залежності від ступеня схожості, усереднити оцінки, які вони вже ставили даному об'єкту [11, 12]. Такий підхід називається user-based kNN, і аналогічно йому можна придумати item-based kNN, а саме подивитися, які товари схожі на той товар, для якого нам потрібно спрогнозувати оцінку, і заповнити цю оцінку на основі схожих товарів.

Таблиця 1.1. Матриця «Користувачі та об'єкти»

	Пила	Зоряні війни	Ванільне небо	1+1
Маша	5	4	1	2
Юля		5	2	
Андрій			3	5

Інший підхід - це матричні розкладання. У цій ситуації ми висуваємо гіпотезу, що оцінку, яку користувач ставить об'єкту, можна наблизити простою моделлю, а саме ми можемо якимось чином підібрати числа, що описують кожен об'єкт, і числа, що описують кожного користувача, отримавши таким чином якісь вектори в просторі однієї і тієї ж розмірності. При цьому вимагати, щоб скалярний добуток вектора, що описує користувача, і вектора, що описує об'єкт, добре наближало оцінку, яку користувач ставить об'єкту. Наближення формалізуємо за допомогою деякої

оптимізаційної задачі [13,14,15]. Наприклад, можна подивитися на квадрати відхилення, як виходить в SVD-розкладанні, або придумати якусь іншу формалізацію, як буває в NMF-разложениях. Профілі товарів і користувачів, тобто вектори, що описують товари та користувачів, ми підбираємо з цієї оптимізаційної задачі. Таким чином, ми отримуємо опис об'єктів і користувачів, за допомогою якого можемо спрогнозувати і ті значення матриці, які нам поки невідомі, тому що користувач поки не бачив об'єкт.

1.4. Підходи до побудови рекомендаційних систем

Перший підхід називається коллаборативна фільтрація. В рамках цього підходу ми будуємо рекомендації для кожного користувача на підставі переваг схожих користувачів. Тобто з множини U ми вибираємо тих користувачів, які схожі на нашого користувача Ui, далі дивимося, як вони оцінили об'єкти, які наш користувач ще не бачив, і на основі цих оцінок будуємо оцінки для тих самих об'єктів. Далі, рекомендації ми будуємо на основі отриманих оцінок.

Наступний підхід до побудови рекомендацій називається content-based. В рамках цього підходу ми знову працюємо з множиною користувачів і множиною об'єктів, однак сам підхід до побудови рекомендацій дуже сильно відрізняється. В рамках цього підходу нам потрібно представити кожен об'єкт з множини у вигляді вектора ознак. Ознаки залежать від тієї предметної області, в якій ми працюємо. Скажімо, якщо ми з вами працюємо з рекомендаційної системою для кіно, то нам потрібні ознаки, які дозволяють описати кожен фільм. Про кожен фільм ми, швидше за все, знаємо жанр, знаємо режисера, знаємо рік видання, знаємо акторів, які в ньому грали. Якщо ж ми говоримо про рекомендаційні системи, пов'язаної з деякими продуктами, наприклад, з одягом, то тут ситуація зовсім інша. Ми можемо

описувати бренд, кольори, розміри, категорію - зовсім інші ознаки, релевантні нашої предметної області [17].

Наступним кроком потрібно описати користувача, адже насправді про користувача ми теж знаємо досить багато інформації. Як мінімум, ми розуміємо, як наші користувачі взаємодіяли з об’єктами. Відповідно, для кожного користувача ми можемо розрахувати ознаки, пов'язані з його поведінкою, наприклад, улюблена категорія продуктів, улюблені фільми, улюблені жанри, улюблені режисери. Всі ці ознаки також допоможуть нам в побудові рекомендаційної системи. Також можна розрахувати ознакиі для кожної пари користувач-об'єкт, що описують їх взаємодію, наприклад, розглянемо рекомендаційну систему продуктів. У цьому випадку ми можемо розрахувати наступні ознаки: переглядав чи раніше користувач цей продукт, як часто користувач купував такі продукти, дата останньої покупки користувачем цього продукту і частота покупок продуктів з цієї ж категорії. Тепер ми можемо зібрати навчальну вибірку, що описує кожну пару користувач-об'єкт. А прогнозувати будемо оцінки, які користувачі дають об'єктам. Таким чином, отримавши навчальну вибірку, ми отримали можливість побудувати модель, модель класифікації або регресії, яка належатиме оцінки [21]. Далі, застосовуючи отриману модель до невідомим парам користувач-об'єкт, ми будемо наближати ті оцінки, які б користувач поставив цьому об'єкту. І, відповідно, на основі таких відновлених оцінок ми будемо будувати рекомендації.

Третій підхід - це рекомендації на основі демографії. Перше, що нам потрібно зробити, це зібрати всі дані, які нам доступні про користувача. Як мінімум, нам доступні дані про те, як користувач використовує нашу систему, але в цілому нам можуть бути доступні і зовнішні, додаткові дані. Наприклад, нам можуть бути відомі соціально-демографічні характеристики користувача: його стать, вік, рід діяльності, освіта. На основі зібраних даних нам потрібно зробити сегментацію користувачів, тобто поділити користувачів на групи схожих користувачів, а далі для отриманих груп

порахувати їх переваги, тобто подивитися, над якими об’єктами мають перевагу користувачі з однієї групи. Коли до нас приходить новий користувач, ми просто дивимося, в яку з цих груп він потрапляє, на яку групу людей він найбільш схожий, і далі ми маємо можливість будувати для нього рекомендації, виходячи з того, які переваги має група [23]. Ну і аналогічним чином ми можемо робити рекомендації для членів групи, виходячи з того, що подобається іншим членам цієї ж групи. Все досить просто. Відповідно, такі рекомендації носять назву demographic-based, засновані на демографії або якихось інших ознаках, що описують користувачів.

Останній підхід називається utility-based. Він дещо схожий на content-based підхід з тієї точки зору, що нам знову потрібно уявити кожен об'єкт за допомогою вектора ознак, але далі починаються розбіжності. Для кожного користувача ми повинні розробити його власну utility function, або функцію корисності, яка буде оцінювати корисність даного об'єкта для даного користувача. Найбільша проблема цього підходу, найголовніше питання, на який потрібно відповісти, як побудувати user-based utility function, тобто як побудувати ось цю функцію для кожного користувача. Припустимо, що ми розробляємо рекомендаційну систему фільмів. Як в цьому випадку може виглядати функція корисності? Ми можемо припустити, що кожен користувач повідомляє нам про те, які інтереси у нього є, тобто які жанри йому подобаються. З одного боку, він може зробити це явно, вибравши ті жанри, які він вважає за краще, з іншого боку, ми можемо зрозуміти це, оцінивши ті фільми, які він переглянув і яким він поставив високі оцінки. Як в такому випадку визначити utility function? Про кожен новий об'єкт, про кожен фільм ми знаємо, який у нього жанр. Давайте побудуємо функцію, яка на підставі жанру фільму буде оцінювати його корисність для даного користувача [30]. Припустимо, якщо фільм має жанр science fiction і користувач любить цей жанр, то корисність цього фільму, значення utility function на ньому буде висока. Якщо ж користувач віддає перевагу інші жанри, наприклад, художні фільми або історичне кіно, то тоді, звичайно,

корисність цього фільму буде менше. Відповідно, таку utility function потрібно розрахувати для кожного користувача і далі на її основі будувати рекомендації, вибирати тільки ті об'єкти, які отримують велике значення по цій функції.

Насправді, виділяють два основних підходи: це коллаборативна фільтрація і content-based рекомендації. Всі інші підходи так чи інакше схожі на якийсь з перших двох і часто розглядаються як їх підмножина. Наприклад, рекомендації на основі демографії дуже схожі на коллаборативного фільтрацію, там використовуються схожі ідеї, тому їх не завжди виділяють в окремий клас, а рекомендації на основі функцій корисності часто розглядають як підмножина content-based рекомендацій. Проведемо порівняльний аналіз цих двох типів рекомендацій.

Спочатку подивимося на коллаборативного фільтрацію. Такий підхід дозволяє нам враховувати крос-жанрові інтереси користувача, тобто будувати рекомендації з різних груп, наприклад, розглядати фільми різних жанрів, продукти різних категорій, ну і взагалі рекомендувати йому товари, не схожі один на одного [25]. З іншого боку, такий тип рекомендацій дозволяє нам використовувати так званий неявний feedback, тобто ми можемо враховувати неявну інформацію, яку користувач залишає про продукти. Це необов'язково повинна бути явна оцінка, ми можемо також використовувати такі сигнали, як перегляд сторінок або час перегляду сторінки деякого об'єкту. Ще один плюс цих рекомендацій полягає в тому, що їх якість поліпшується з часом. Чим більше статистики про кожного користувача і про кожен об'єкт ми накопичили, тим більше інформації ми знаємо, і тим більше якісні рекомендації ми отримуємо. Тепер давайте подивимося, які ж у цього методу є проблеми. Ну, по-перше, в рамках цього методу дуже сильно актуальна проблема холодного старту, причому вона актуальна як для користувачів, так і для об'єктів. Коли ми бачимо нового користувача, він ще не встиг повзаємодіяти з нашою системою і не встиг оцінити багато об'єктів. Відповідно, у нас з вами недостатньо інформації для

того, щоб зробити для нього хороші рекомендації, щоб зрозуміти, на кого він схожий і які об'єкти йому потрібно рекомендувати. З іншого боку, коли у нас з'явився новий об'єкт, не дуже багато користувачів встигли його оцінити, і нам знову складно його комусь рекомендувати [33]. Також в рамках цього підходу актуальна проблема gray sheep, або сірої вівці. Ця проблема полягає в тому, що якщо у нас є користувачі, які не схожі на інших, наприклад, користувач, у якого просто немає близьких користувачів в нашій базі даних, то ми нічого не можемо йому порекомендувати, тому що він ні на кого не схожий, ми не зможемо поширити на нього реакцію інших користувачів. Ну і також завжди актуальна проблема популярних об'єктів. Якщо у нас є об'єкт, який всі подивилися, який всім сподобався, то, звичайно ж, ми будемо постійно рекомендувати його для всіх. Це досить очевидна рекомендація, користі від якої не дуже багато.

В рамках content-based підходу ми маємо майже ті ж самі переваги. Ми знову можемо використовувати неявний feedback користувача, і якість нашої системи буде тільки зростати з часом. Однак нам вже не так просто будувати крос-жанрові рекомендації. З іншого боку, у нас набагато менше проблем. Як мінімум, ми позбавляємося від проблеми холодного старту для нових об'єктів, адже як тільки об'єкт з'являється в нашій системі, ми відразу ж розуміємо, які у нього є ознаки, а, значить, нарівні з усіма іншими об'єктами можемо рекомендувати його користувачам. З іншого боку, проблема «сірої вівці» також вирішується, тому що ми набагато менше покладаємося на те, наскільки наш користувач схожий на всіх інших, ну і проблема популярних товарів теж актуальна з дуже меншою мірою.

1.5. Проблема в кінематографії

Моделі використовуються всюди, на будь-яких підприємствах та у будь-яких організаціях. Десь вони вже пройшли стадію реалізації в готові

програмні продукти, а десь використовуються неявно, лише за допомогою працівників закладів.

Попит породжує пропозицію, і різноманітність програмних продуктів з кожним роком зростає. Але далеко не всюди є можливість включити готове рішення, іноді для його впровадження необхідна деяка реорганізація підприємства.

В кінематографії наразі якраз той випадок, де готові рішення не мають змоги бути реалізованими «з коробки» через специфічність внутрішньої організації роботи підприємства.

Основною причиною є те, що коли новий фільм виходить в прокат, інформації про взаємодію цього фільму з клієнтами ще немає для того, щоб використати готові алгоритми які працюють виключно з інформацією про взаємодію клієнта-об’єкта.

1.6. Огляд існуючих програмних продуктів

Серед існуючих реалізацій програмного забезпечення які хоча б частково вирішували поставлену проблему є програми «Einstein» від Salesforce та AutoML від Google.

Ці програми створені для того щоб полегшити розробку рекомендаційних систем, однак жоден з існуючих програмних продуктів не дозволяє повної автоматизації та масштабування рекомендаційної системи.

Программа «Einstein» представляється як готовий сервіс для рішення задач рекомендаційних систем, прогнозування продажів, роботи з відео та обробкою природньої мови.

Серед переваг цього сервісу слід відзначити наступні:

- Готова система, що легко масштабується на будь-який мікросервіс, має модульну структуру.
- Легка в обслуговуванні
- Має внутрішню базу даних яка включає в себе інформацію про фільми.

Серед недоліків основним є те що користувач сам складає правила, за якими мають створюватися рекомендації, що займає дуже багато часу. Також система ніяким чином не використовує дані про взаємодію клієнта з об’єктами.

Сервіс AutoML є більш цікавим інструментом для створення рекомендаційної системи. Окрім функцій, вказаних у «Einstein», в AutoML реалізована можливість авто-створення пуш-повідомлень з красивим графічним оформленням, багато можливостей по управлінню даними, тощо. Переваги цього сервісу більш вагомі:

- Можливість попереднього формування пуш-повідомлень
- Велика кількість налаштувань
- Автоматично оновлювана інформація бази даних фільмів
- Візуально приємний інтерфейс
- Модульність системи

Однак створення рекомендацій для нових фільмів, з якими не було взаємодії в повній мірі не вирішується жодною із систем.

Оцінка самого фільму, пошук попередніх рейтингів, намагання подумати за глядача та передбачити який саме фільм порекомендувати щоб прибуток був більший – на такі запити немає відповіді в жодному з аналогів.

Саме тому метою роботи стало створення програмного продукту, який відповів би на поставлені питання.

1.7. Особливості поставленої задачі

Принцип створення рекомендацій доволі простий - розташувати фільми в пуш-повідомленні таим чином, щоб на першому місці був фільм, який вірогідніше всього зацікавить клієнта. Це доволі просто зробити коли ми маємо історію взаємодії клієнта з різними об’єктами та іторію взаємодії об’єкту з клієнтами, але це перетворюється на складну задачу коли

рекомендації потрібно створювати в кінотеатрах, де з прем’єрними фільмами не було ніяких взаємодій.

Для того щоб мати уявлення про те як клієнти будуть взаємодіяти з новим фільмом – потрібно знати хоча б частково інформацію про фільм, для того щоб віднеси фільм до якогось із кластерів.

Також потрібно врахувати дані клієнта про його уподобання та створити рекомендації для тих клієнтів, які мають невелику історію транзакцій.

Відсутність єдиної бази фільмів створює складнощі – у різних кінотеатрах фільм може називатися по різному, може відрізнятись прокатна інформація, дозволений вік для перегляду, тощо.

Відсутність параметрів які б могли в повній мірі описати характеристики фільма для його подальшого аналізу.

Також щодо вікових обмежень (таблиця 1.2), то вони значно відрізняються від міжнародного стандарту, їх більше та вони часто пересікаються між собою, що не дає змоги однозначно класифікувати фільм.

Таблиця 1.2. Порівняння систем обмежень

Американська система	Українська система
G	0+, 1+, 2+, 3+
PG	6+, 7+
PG-13	12+, 14+
R	18+, 21+

1.8. Моделі прогнозування та кластеризації

Система прогнозування – впорядкована сукупність методик, технічних засобів, що призначена для прогнозування складних явищ та процесів. [9]

Математична модель це або фізичне представлення математичних понять, або математичне представлення реальності. [10]. Математична модель як фізичне представлення включає в себе репродукції твердих тіл з картону, дерева, пластику, чи інших матеріалів, і не є об'єктом нашого дослідження. Набагато ширшим є поняття математичної моделі як способу представлення реальності через формули і математичні поняття. В цілому, будь що в фізичному чи біологічному світі, не важливо, природне воно чи створене з впливом технологій та людини, може бути проаналізоване за допомогою математичних моделей, якщо його можна описати в математичних термінах [10]. Моделі предметної області – математичні моделі прогнозування, для створення яких використовуються деякі відомі законі обраної предметної області. Для розробки таких моделей потрібен індивідуальний підхід та спеціаліст з обраної області [11]. Моделі часових рядів – моделі, за допомогою яких можна знайти залежність майбутніх значень від минулих всередині процесу, і через цю залежність порахувати прогноз. Такі моделі є універсальними, їх зовнішній вигляд не змінюється від природи часового ряду [11].

Що до класифікації моделей (рис. 1.1.), то в моделях предметної області таке представлення не є можливим – скільки областей стільки і моделей. На відміну від них, моделі часових рядів можна розділити на статистичні та структурні моделі [11].

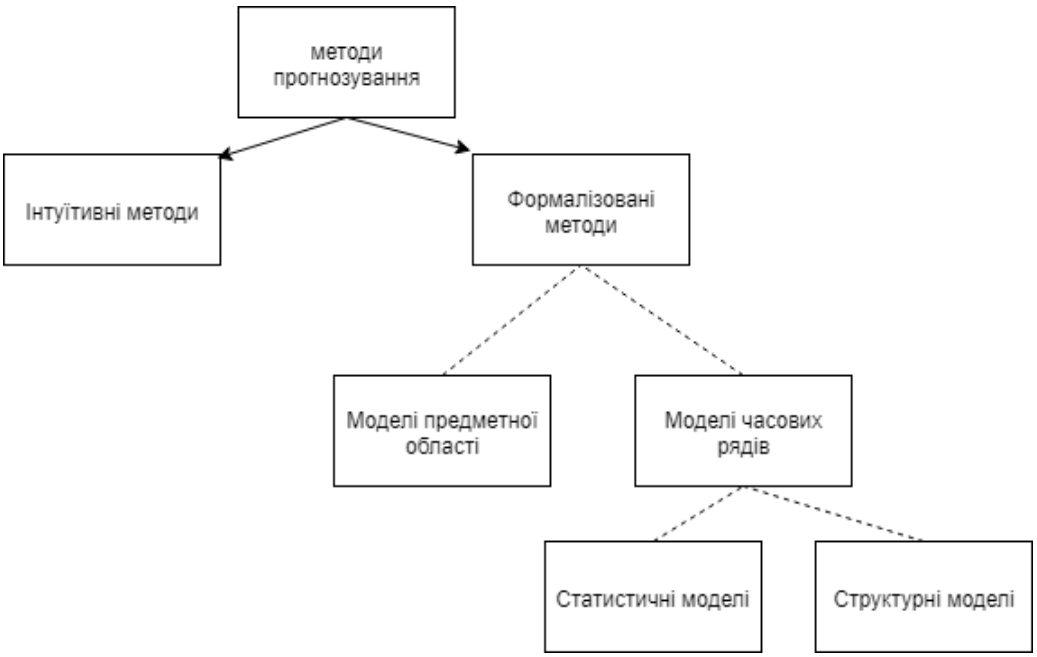


Рисунок 1.1. Класифікація методів і моделей прогнозування

В статистичних моделях залежність минулого від майбутнього задається у вигляді рівняння [11]. До них відносяться такі як :

- Регресійні моделі
- Авторегресійні моделі
- Моделі експоненційного згладжування (модель Холта-Вінтерса, модель Брауна).

В структурних моделях залежність майбутнього значення від минулого задається у вигляді певної структури та правил переходу по ній [11].
Наприклад:

- Моделі на базі нейронних мереж
- Моделі на базі ланцюгів Маркова
- Моделі на базі класифікаційно-регресійних дерев

В роботі використано модель на базі нейронних мереж, а саме модель Embeddings.

Моделі кластеризації (рис 1.2.) - багатовимірні статистичні процедури, що впорядковують об'єкти в порівняно однорідні групи [1] [2] [3] [4].

Завдання кластеризації відноситься до статистичної обробки, а також до широкого класу задач навчання без учителя.

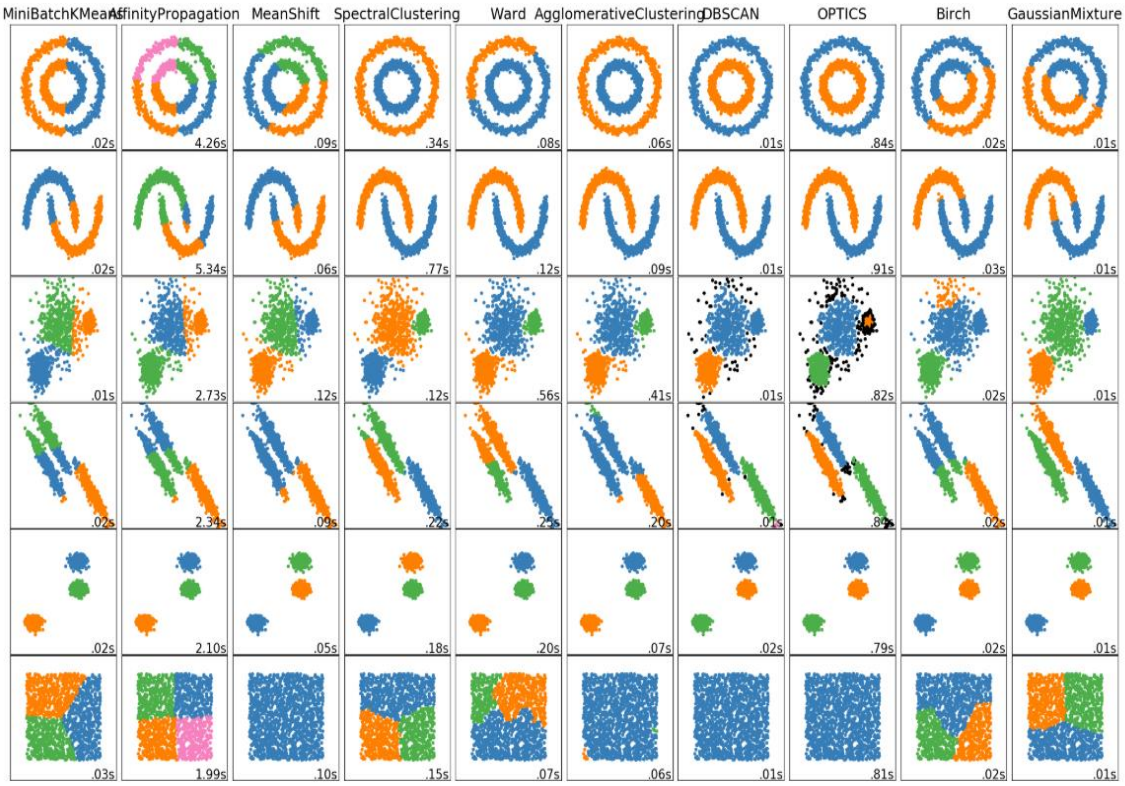


Рисунок 1.2. Класифікація методів і моделей кластеризації

ВИСНОВКИ ДО РОЗДІЛУ 1

В даному розділі було розглянуто поняття математичної моделі, моделювання, наведено підходи до побудови рекомендаційних систем, їх відмінності, та проблеми які виникають в кожному із типів. Приведено класифікацію математичних моделей і методів, визначено до якого класу методів відноситься вибраний для роботи алгоритм.

Проаналізовано зовнішнє оточення проекту, виявлено відсутність аналогів продукту.

Розглянуто основні вимоги до програмної реалізації модульних систем. Окрім того, наведено стислу історію виникнення математичного моделювання.

В результаті можна зробити висновок, що математичне моделювання це потужний інструмент для аналізу складних систем який можна використати для побудови рекомендаційної системи за допомогою якої можна реалізувати пуш-повідомлення з прем’єрними показами в кінотеатрі, тестуючи ідеї у оточенні, програмно наближеному до реального.

РОЗДІЛ 2
АНАЛІЗ ВХІДНИХ ДАНИХ ТА ПОБУДОВА ГІПОТЕЗ

2.1 Визначення основних гіпотез поведінки клієнта

Після того як дані були зібрані та очищені було проведено детальний аналіз структури даних, поведінкових шаблонів та основні фактори впливу, які будуть використовуватися в моделюванні.

Початкова гіпотеза – вибір фільму залежить від дня тижня, в який він буде трансльований. Проілюструємо це за допомогою гістограми (рис. 2.1.).

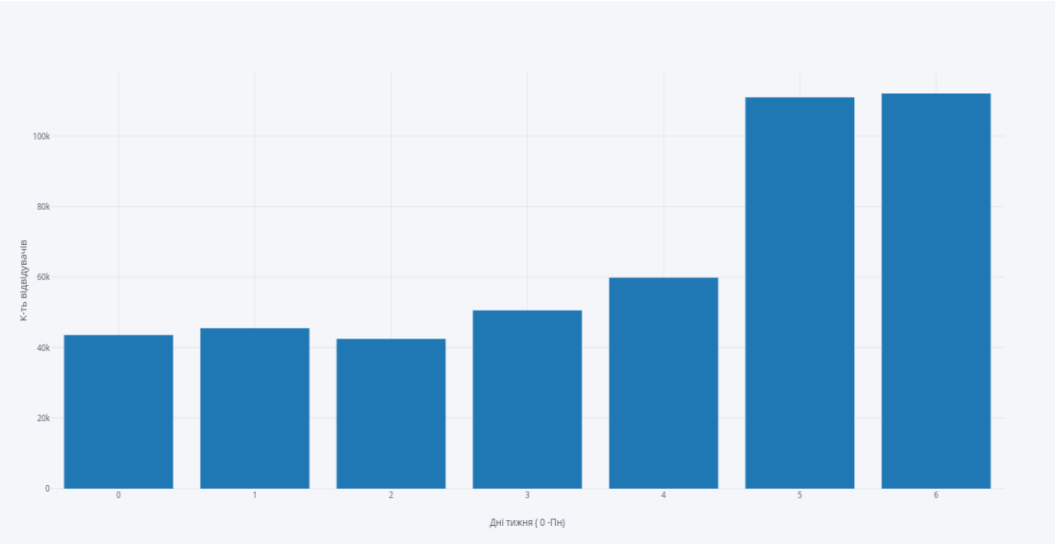


Рис 2.1. Кількість куплених білетів в залежності від дня тижня

Гістограма показує, що в вихідні дні (5-6) значно зростають продажі білетів порівняно з робочими днями (0-4). Для опису такої поведінки доцільно ввести ознаку, яка вказує на те чи це робочий день чи вихідний. Така ознака може покращити рекомендаційні прогнози завдяки влучному прогнозуванню відносно дня тижня.

Друга гіпотеза, яку було розглянуто - жанрова зацікавленість клієнтів. Щоб візуалізувати уподобання клієнтів відносно жанрів була проаналізована популярність жанрів в фільмах, на які були куплені білети. (рис. 2.2.)

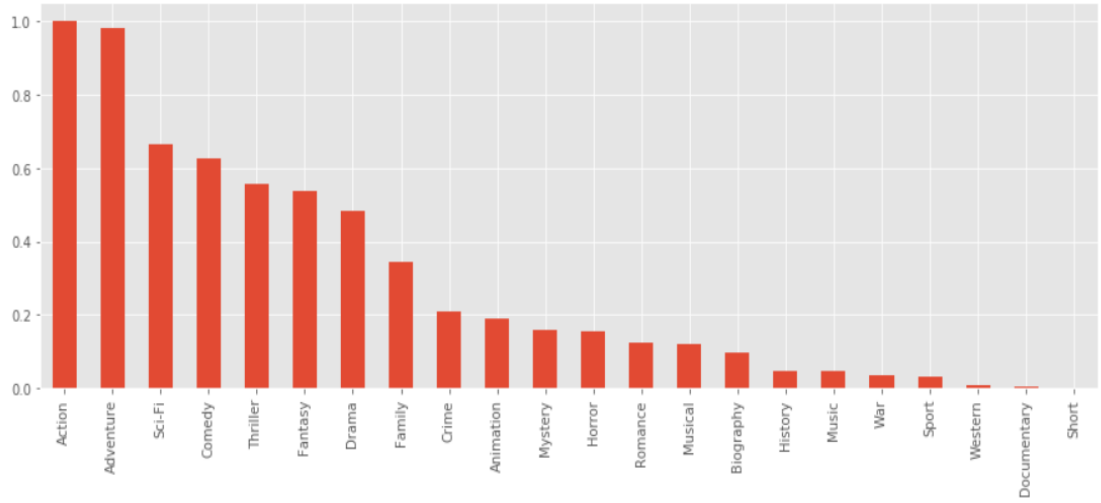


Рис. 2.2. Найпопулярніші жанри фільмів які були показані в кінотеатрі

Як можна побачити з наведеної вище гістограми (рис. 2.2) фільми з жанрами Action, Adventure найпопулярніші, що говорить про більшу зацікавленість саме в таких фільмах. Отже можна зробити висновок, що на рекомендації суттєво впливає жанр фільму.

Розглянемо відвідування клієнтів в розрізі 1 року (рис 2.3.).

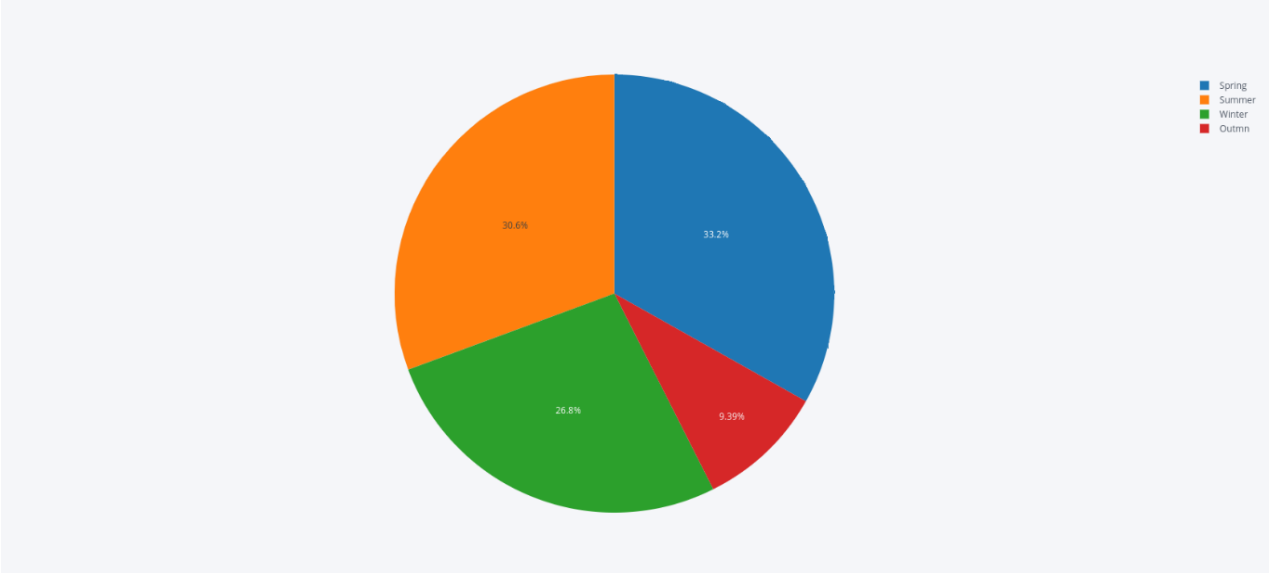


Рис. 2.3. Відвідування за 1 рік в розрізі сезонів

На круговій діаграмі видно, що зимою та весною показник відвідування найвищий і варіюється близько 31%, а осінні показники відвідування в свою

чергу рівні лише 9%. Показник сезонності може також впливати на уподобання клієнтів.

Для побудови системи потрібно також побудувати модель, яка буде кластеризувати фільми, для того щоб для нового фільму знаходити фільми, які схожі на нього. Для кластеризації фільмів було розглянуто гіпотезу подібності фільмів на жанровому рівні (рис. 2.4.).

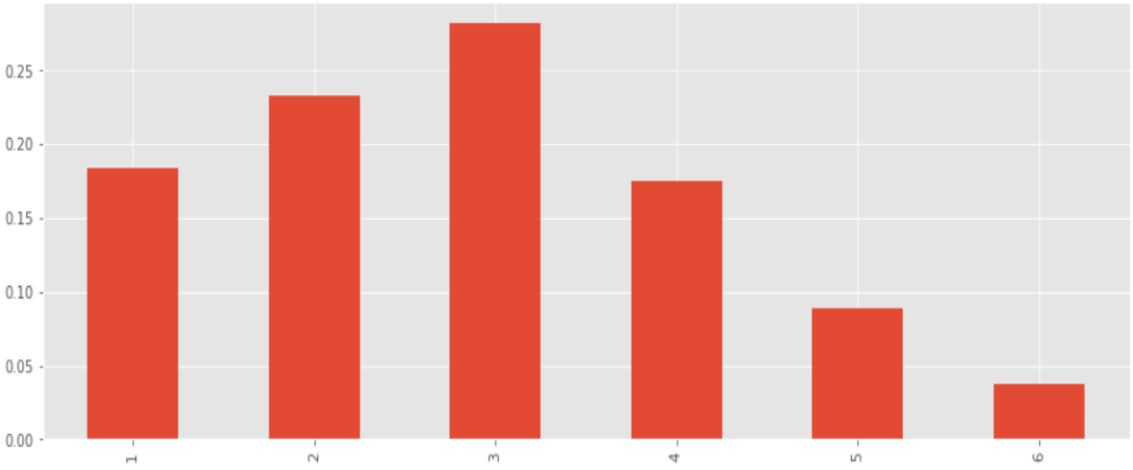


Рис. 2.4. Відсоткове відношення кількості жанрів в розрізі фільмів

Як видно з гістограми (рис 2.4.) кількість жанрів для одного фільму може бути від одного до шести, але основна маса фільмів має від 2 до 4 жанрів, що означає достатню кількість даних для побудови моделі.

Наступною гіпотезою подібності фільмів є схожість по ключовим словам, які притаманні фільму. (рис. 2.5.)

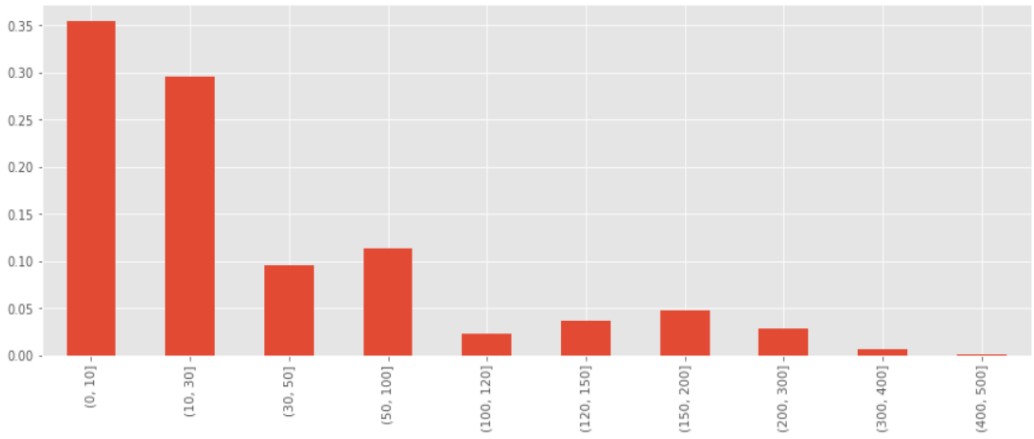


Рис. 2.5. Відсоткове відношення кількості ключових слів у фільмах

Як видно з гістограми (рис 2.5.) близько половини фільмів мають більше ніж 100 ключових слів. Ця кількість є великою для опрацювання алгоритмом, тому потрібно фільтрацію ключових слів які зустрічаються вкрай рідко або ж є стоп словами.

Ще однією гіпотезою є те, що фільми створені в схожих країнах також схожі (рис. 2.6.)

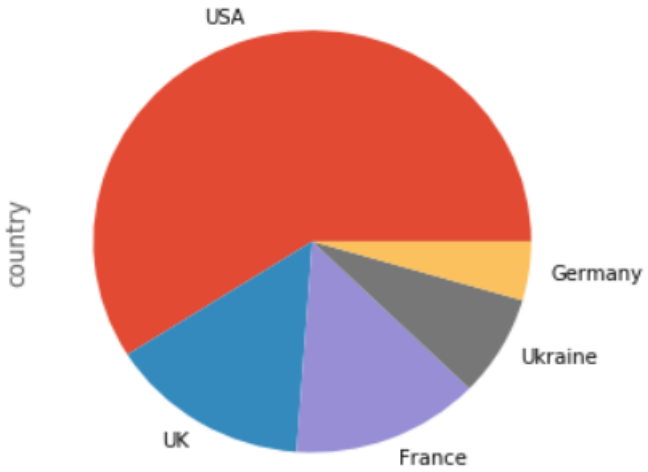


Рис. 2.6. Кругова діаграма розподілу фільмів відносно країни

Як видно з кругової діаграми (рис 2.6.) 63% фільмів були зняті в US, інша частина в європі. Також видно, що кількість унікальних значень невелика, що полегшує використання даної ознаки.

Також була розглянута гіпотеза, яка стверджує, що в одного режисера фільми схожі між собою та мають притаманний режисеру стиль. (рис. 2.7.)

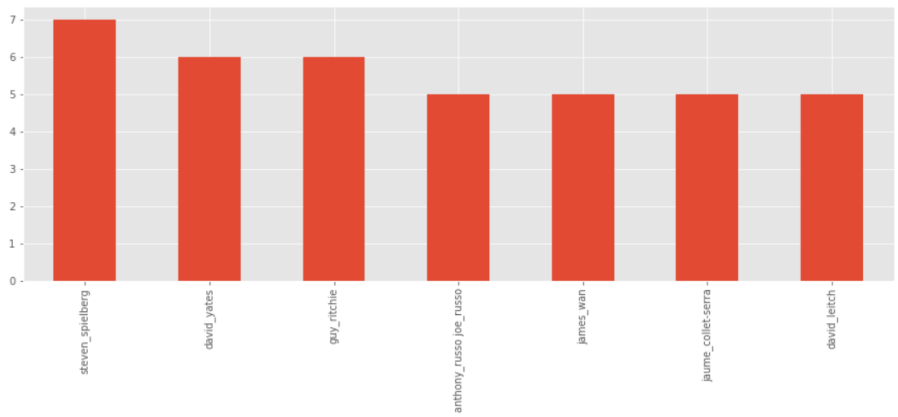


Рис. 2.7. Гістограма кількості фільмів в одного режисера

На гістограмі (рис 2.7.) показано, що в відомих режисерів є декілька фільмів, які можуть бути схожі між собою. Отже клієнти скоріше за все підуть на фільм, який знятий їх улюбленим режисером, ніж на інший фільм.

Останньою гіпотезою було розглянуто подібність акторів, знятих у фільмі (рис. 2.8.)

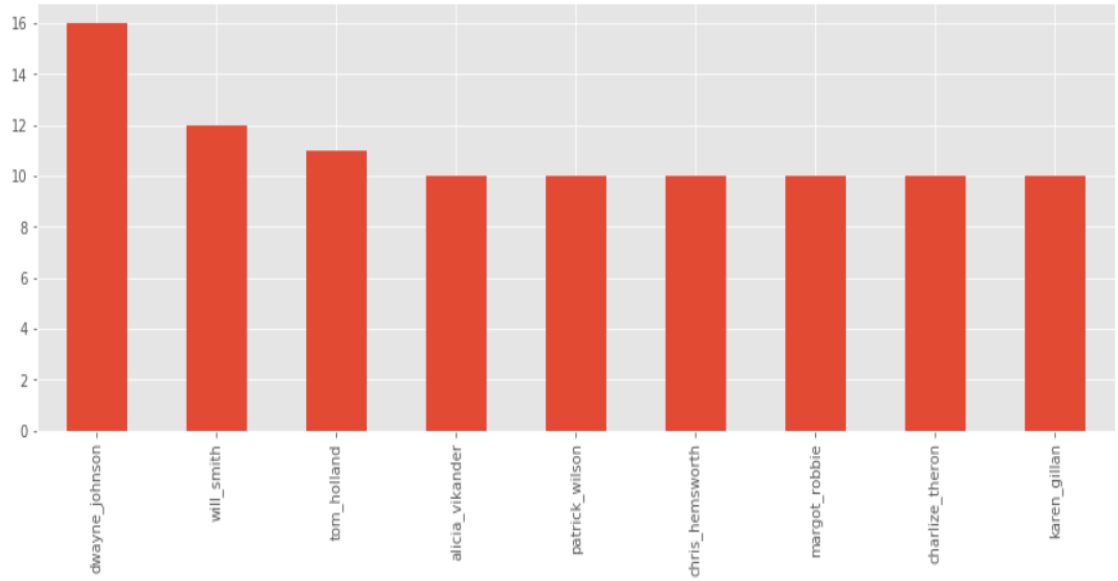


Рис. 2.8. Гістограма кількості фільмів, в якому грає актор

На гістограмі (рис 2.8.) показано, що існує велика кількість фільмів, де грають одні і ті ж актори. Отже клієнти скоріше за все підуть на фільм, в якому грає їх кумир, ніж на інший фільм.

Також було перевірено розподіл фільмів відносно року релізу (рис. 2.9.).

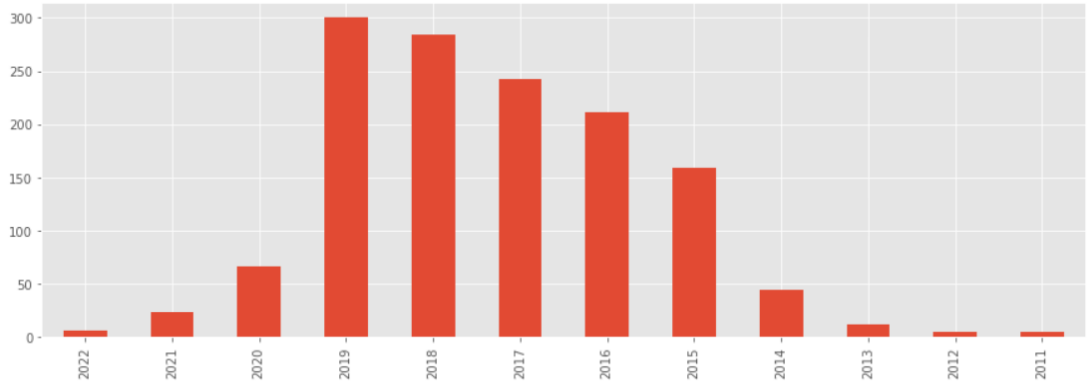


Рис. 2.9. Гістограма кількості фільмів в розрізі року

Можна зробити висновок що база фільмів включає в себе тільки фільми які були зроблені відносно недавно.

На бар-плоті (рис 2.10.) видно, циклічність продажів білетів протягом року. Період для аналіза вибраний в 6 місяців

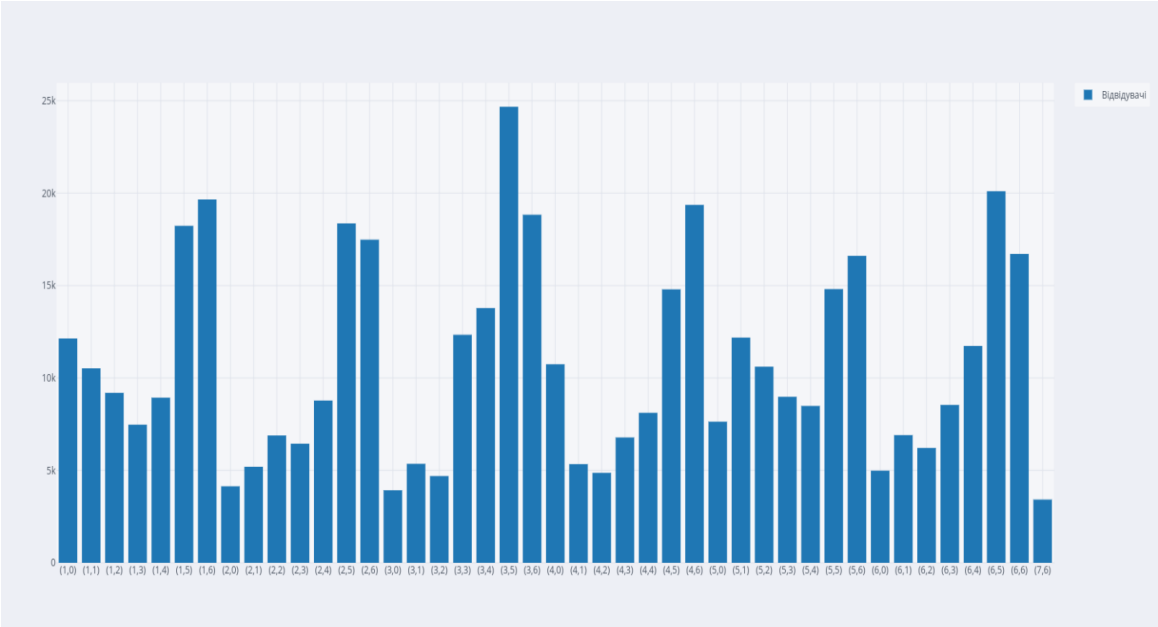


Рис. 2.10. Бар-плот циклічності даних

Також було перевірено вплив технології на вибір фільму. Наприклад деякі клієнти більше передають перевагу фільмам в IMAX. (рисунок 2.11).

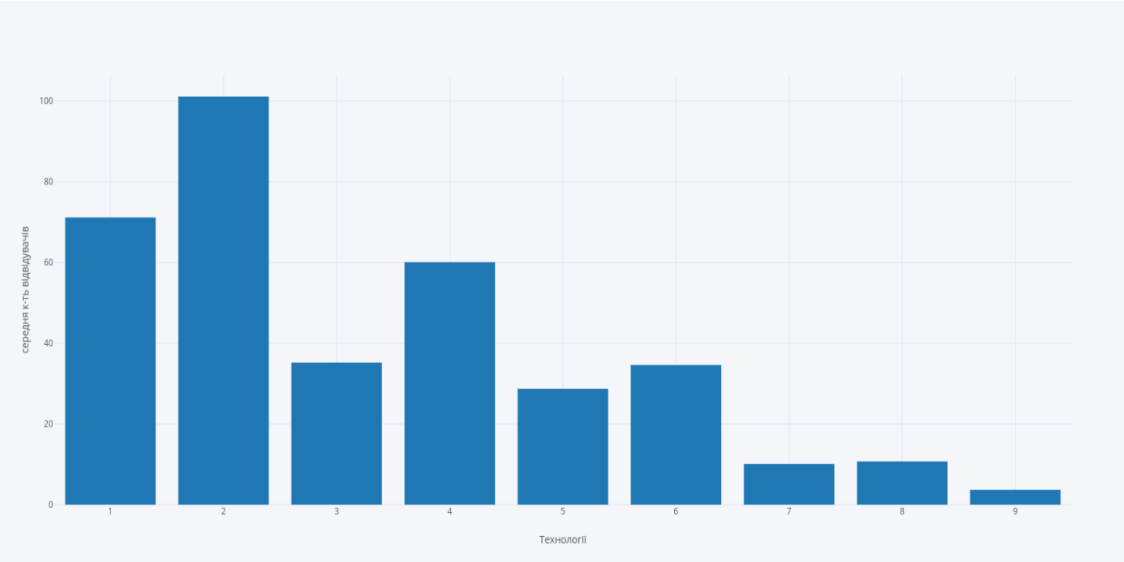


Рис. 2.11. Середня кількість відвідувань в розрізі технології

Ще однією гіпотезою було те, що час проведення фільмів був різним, що могло вплинути на його вибір. (рисунок 2.12).

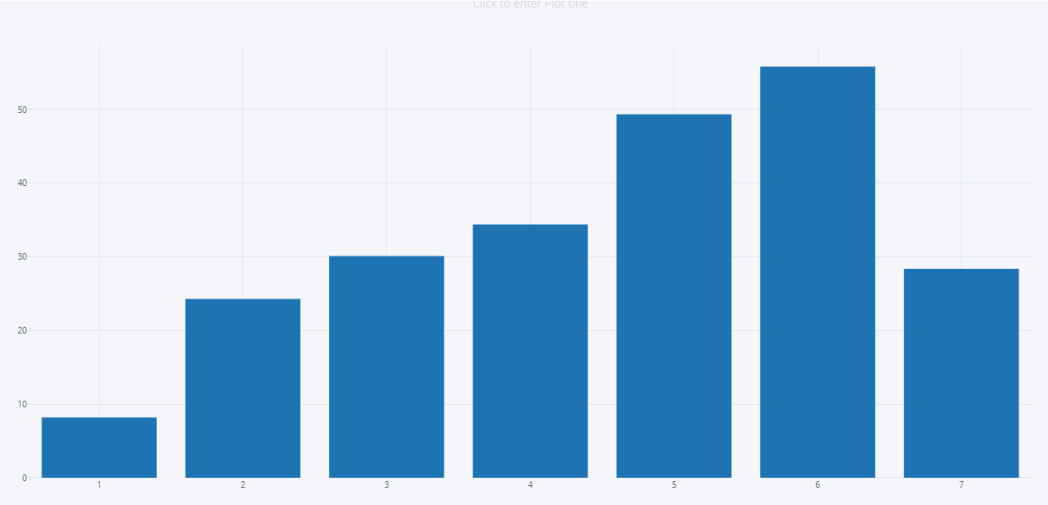


Рис. 2.12. Середня кількість відвідувань в розрізі години дня

2.2 Формування груп схожих між собою фільмів

Для збору даних використовується веб-сервіс, який віддає запити на API OMDB для отримання або оновлення інформації.

Для вирішення проблеми було поділено фільми за віковими обмеженнями а потім кластеризовано. OMDB дозволяє отримати вікові обмеження фільмів в наступному вигляді (таблиця 2.1.):

Таблиця 2.1. Код вікових обмежень

Код	Ознака
G	Мультики для дітей
PG	Мультики для підлітків
PG-13	Фільми для підлітків
R	Фільми для дорослих

Для створення рекомендацій з нових фільмів нам потрібно знати аналоги таких фільмів, щоб в першу чергу рекомендувати клієнту той фільм, який має найбільше переглянутих ним аналогів.

Після реалізації отриманих гіпотез, було вирішено спробувати кластеризувати фільми за жанрами за допомогою методу мішка слів. Але в ході аналізу результатів було виявлено наступні недоліки:

- Через прокляття розмірності алгоритм кластеризації не може розділити кластери на малі частини.
- Фільми з віковими обмеженнями потрапляють в кластер з мультиками.

Для побудови системи було вирішено розділити фільми на дві підгрупи, які включали б в себе вікові обмеження (G, PG) та (PG-13, R).

Також було досліджено, що фільми, вироблені в різних країнах, дуже відрізняються один від одного, тому було вирішено також поділити фільми на групи:

1) USA
2) Europe
3) Ukraine or Russia

Разом з країною виробника також використовувався жорсткий поділ фільмів за деякими режисерами. Якщо в обох фільмів спільний режисер, то ці два фільми автоматично рахуються ідентичними.

Приклади відомих режисерів, які мають однаковий стиль:

- Christopher Nolan
- Wes Anderson
- Quentin Tarantino
- Woody Allen
- James Cameron
- David Fincher
- Guy Ritchie

У фільмів також наведені ключові слова, які було використано разом з жанрами. Всі наведені методики значно покращили результат підбору схожих фільмів.

2.3 Механізм створення рекомендацій

Для реалізації рекомендаційної системи фільмів з холодним стартом було використано нейронну мережу. яка створює два ембедінги опису фільмів ключовими словами, і навпаки, для того щоб отримати матрицю подібності фільмів між собою. Модель системи було поділено на три частини (рис. 2.13.). Перша частина виконує збір та валідацію даних.

Друга частина використовує дані, що описують фільм для навчання моделі ембедингів, щоб порахувати матрицю схожості фільму до інших фільмів. Це дає змогу знайти аналоги для нових фільмів, які були в прокаті раніше і мають історію переглядів іншими користувачами кінотеатру. Вибір алгоритму пошуку фільмів за допомогою знаходження описових ембедінгів було взято після тестування різних підходів кластеризації. Найкращий метод знаходження коефіцієнтів схожості фільмів побудований на алгоритмах обробки природної мови який з текстових описів фільму ,які включають в себе ключові слова та жанри, найкраще виділяє схожі ознаки.

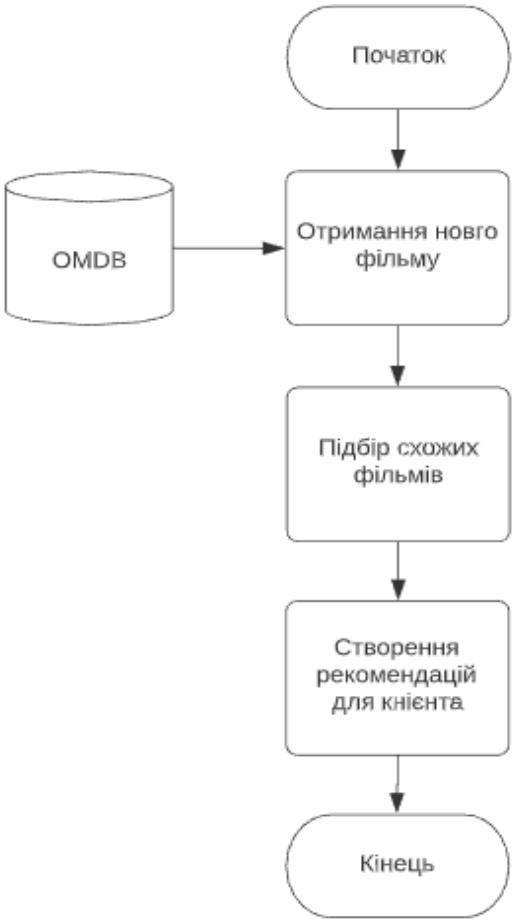


Рис. 2.13. Модель створеної системи

Третя частина формує для кожного клієнта список нових фільмів відсортованих за відсотковою ймовірністю того, що фільм сподобається клієнту. Також в цій частині використовуються мета-інформація клієнта та фільма, що дозволяє уникнути рекомендації фільмів для дорослих дітям. Також знаючи місто проживання клієнта ми можемо рекомендувати тільки ті фільми, які йдуть в прокаті в даному місті.

Отже ці три частини складають систему, яка реалізована на мові програмування Python та випробувана на реальних даних однієї з мереж кінотеатрів України.

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі було проведено детальний огляд роботи кінематографії, розглянуто гіпотези щодо опису поведінкових залежностей в моделюванні для покращення прогнозних результатів, розуміння уподобань клієнтів, створення механізму рекомендацій.

Була проаналізована мета-інформація про клієнта. Тестування різних алгоритмів пошуку схожих фільмів показало що деякі алгоритми на такому класі задач працюють значно гірше ніж на інших задачах, тому було вирішено перейти до більш складних алгоритмів які базуються на алгоритмах які використовуються в обробці природної мови, для того щоб краще виділити спільні характеристики фільмів, що в подальшому покращило б результати пошуку релевантних фільмів для клієнта.

Архітектура створення рекомендацій включає в себе обробку фільмів з використанням мета-інформації про клієнта.

Мета-інформація про фільм також має суттєвий вплив на пошук коефіцієнтів релевантності, для цього використовуються додаткові описи фільмів де можна знайти спільних акторів чи режисерів, що допоможе в подальшому влучно розподілити фільми на схожі підгрупи.

Основною задачею було зрозуміти поведінкові залежності клієнтів та проаналізувати різні методи кластеризації фільмів, що в подальшому могло б використовуватися в створенні рекомендацій.

Також була розроблена система збору та оновлення інформації, яка має найнижчий ступінь навантаження. Проведено аналіз зовнішніх джерел збору інформації.

Всю систему також було розділено на підмодулі які б могли легко масштабуватися на різні підзадачі. Наприклад, система пошуку схожих фільмів може бути використана як окремий продукт, що пропонує знайти щось схоже до того, що вам дуже сподобалося.

Розроблено головну модель системи для створення програмного рішення формування рекомендацій щодо перегляду фільмів за допомогою використання інформації. що надається безкоштовно, за допомогою API.

РОЗДІЛ 3
ПРОЕКТУВАННЯ ТА РОЗРОБКА РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ ФІЛЬМІВ З ХОЛОДНИМ СТАРТОМ

3.1. Модуль збору та оновлення даних

Першим етапом роботи системи є збір даних. Для його реалізації використовується сховище даних OMDB (таблиця 3.1.).

Таблиця 3.1. Приклад отриманих даних

Поле	Значення	Тип поля
MovieTitle	Cinderella	string
Array_of_genres	[Family, Fantasy]	array
Age_limit	PG	string
Key_words	[17th century, disney, cinderella]	array
Country	[USA, UK]	array
Director	[Kennetech Brangath]	array
Actors	[Cate Blanchett, Lily James, Richard Madden, Helen Maddison]	array

Збір даних було автоматизовано за допомогою мікросервісу, який має доступ до API. Механізм роботи мікросервісу (рис. 3.1.):

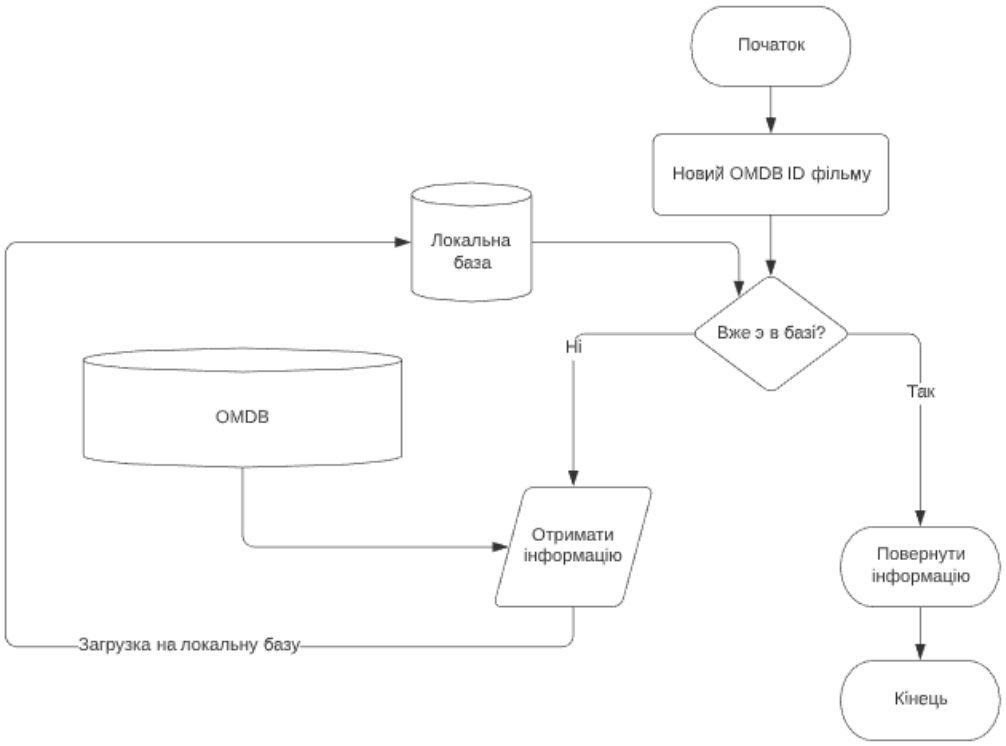


Рис. 3.1. Механізм роботи мікросервісу для збору даних

Користувач подає запит у вигляді POST запиту наступного формату (3.1.):

body: {"imdb_Id":”aa4867312”} (3.1.)

POST запит відправляється до мікросервісу, де перевіряється наявність фільму в локальній базі даних, і якщо фільм вже є в базі, то інформація повертається без використання API OMDB.

У разі якщо фільм відсутній у локальній базі, то мікросервіс виконує збір даних використовуючи зовнішні дані OMDB і записує їх в локальну базу.

Також в мікросервісі реалізоване оновлення даних, яке відбувається з 3 до 5 години ранку, в час коли система має найнижче навантаження.

Для роботи з даними клієнтської бази було створено сховище даних, яке оновлюється з 4 до 5 години ранку, беручи дані з головної бази транзакцій, та головного сховища даних кінотеатру.

Основні таблиці даних з якими ми будемо оперувати - users, likes, ticket_sales.

В таблиці users зберігається вся мета-дата про клієнта: його вік, ім’я, дата народження, дата реєстрації, місто. Вік нам дозволить уникнути ситуацій коли людині рекомендується фільм, вік якої менший за дозволений мінімальний вік для перегляду фільму. Інформація про місто де проживає клієнт дозволить нам рекомендувати тільки ті фільми, прокат яких є в місті проживання.

Таблиця likes дозволить нам зрозуміти які з фільмів клієнт точно хоче подивитися, до того ж інформацію про лайки можна використовувати так, як і про покупку фільму, тобто фільм для якого клієнт поставив лайк рахується як фільм на який клієнт купив би білет.

Дані з таблиці ticket_sales дозволяють отримати список фільмів які клієнт вже подивився, ця інформація потрібна як для створення вектора жанрової зацікавленості клієнта, так і для знаходження коефіцієнтів релевантності.

3.2. Алгоритм пошуку схожих фільмів

Наступним етапом є загрузка даних з створеної локальної бази даних, яка описана в першому етапі, в другу частину алгоритму, де відбувається знаходження списку схожих фільмів. Основним механізмом є нейронна мережа яка навчається методами обробки природної мови на інформації про фільми яка включає в себе ключові слова та жанри. Також виконується розподілення фільмів за їх віковими обмеженнями, країною-виробником та режисером. Схема роботи другого етапу алгоритму (рис. 3.2.):

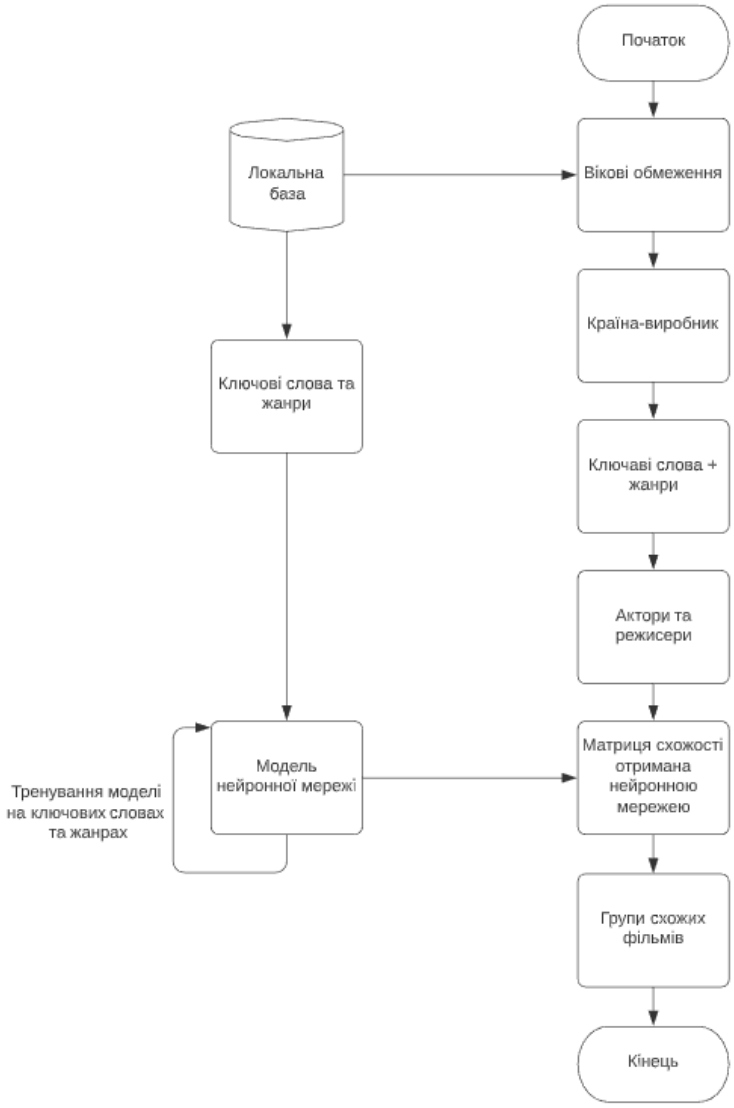


Рис. 3.2. Схема роботи другого етапу алгоритму

Алгоритм пошуку схожих фільмів базується на знаходженні матриці коефіцієнтів схожості яка будується за допомогою навчання ембедингової нейронної мережі (3.1). Для навчання використовуються дані опису фільмів жанрами та ключовими словами. (3.1)

```
def film_embedding_model(film_index, tag_index, embedding_size = 50):
    """Model to embed filns and tags using the functional API"""

    # Both inputs are 1-dimensional
    film = Input(name = 'film', shape = [1])
    tag = Input(name = 'tag', shape = [1])

    # Embedding the film (shape will be (None, 1, 50))
```

```
film_embedding = Embedding(name = 'film_embedding',
                    input_dim = len(film_index),
                    output_dim = embedding_size)(film)

# Embedding the tag (shape will be (None, 1, 50))
tag_embedding = Embedding(name = 'tag_embedding',
                    input_dim = len(tag_index), output_dim = embedding_size)(tag)
# Merge the layers with a dot product along the second axis (shape will be (None, 1, 1))
merged = Dot(name = 'dot_product', normalize = True, axes = 2)([film_embedding, tag_embedding])

# Reshape to be a single number (shape will be (None, 1))
merged = Reshape(target_shape = [1])(merged)


model = Model(inputs = [film, tag], outputs = merged)
model.compile(optimizer = 'Adam', loss = 'mse')
```

3.2.1. Поділ фільмів за віковими обмеженнями

Спершу фільми розподіляються на три категорії {G, PG}, {PG-13} і {R} для того, щоб чітко відділити фільми для дітей від фільмів з віковими обмеженнями, для уникнення ситуацій коли дитині рекомендується фільм 18+.

Підліткам, які входять в вікову категорію PG-13 спочатку рекомендуються фыльми з їх категорії, в разі ж якщо фільмів в цій категорії недостатньо, йому будуть запропоновані фільми з категорії {G, PG}.

Для всіх людей які дорослiші за 18 років спочатку шукаються фільми в категорії для дорослих, потім в категоріях {PG-13} та {G , PG}.

3.2.2. Поділ фільмів за країною-виробником

Потім відбувається сортування фільмів за країною-виробником для розділення фільмів пострадянських країн, країн Європи та Америки (рис. 3.3.), щоб в подальшому людям, які полюбляють пострадянські фільми, рекомендувати їх в першу чергу . Крім того, в деяких країнах є авторське кіно. Також, кіно експертом було зазначено що фільми призначені для країн

Європи та Америки суттєво відрізняються, тому краще щоб такі фільми знаходилися в різних підгрупах.

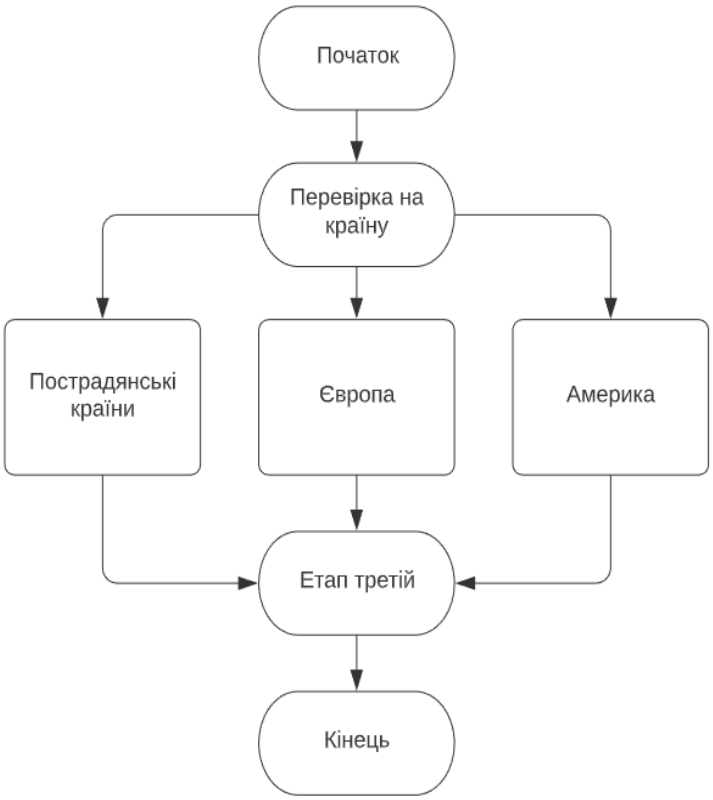


Рис. 3.3. Сортування за країною

3.2.3. Групування фільмів за жанрами та ключовими словами

Наостанок відбувається створення груп фільмів за жанрами або ключовими словами, це допомагає знайти групи фільмів, які мають специфічний сюжет, притаманний тільки даній групі. Такі ключові слова як Disney, Zombie, Spy, World War та інші допомагають краще розділити подібні фільми. (рис. 3.4.).

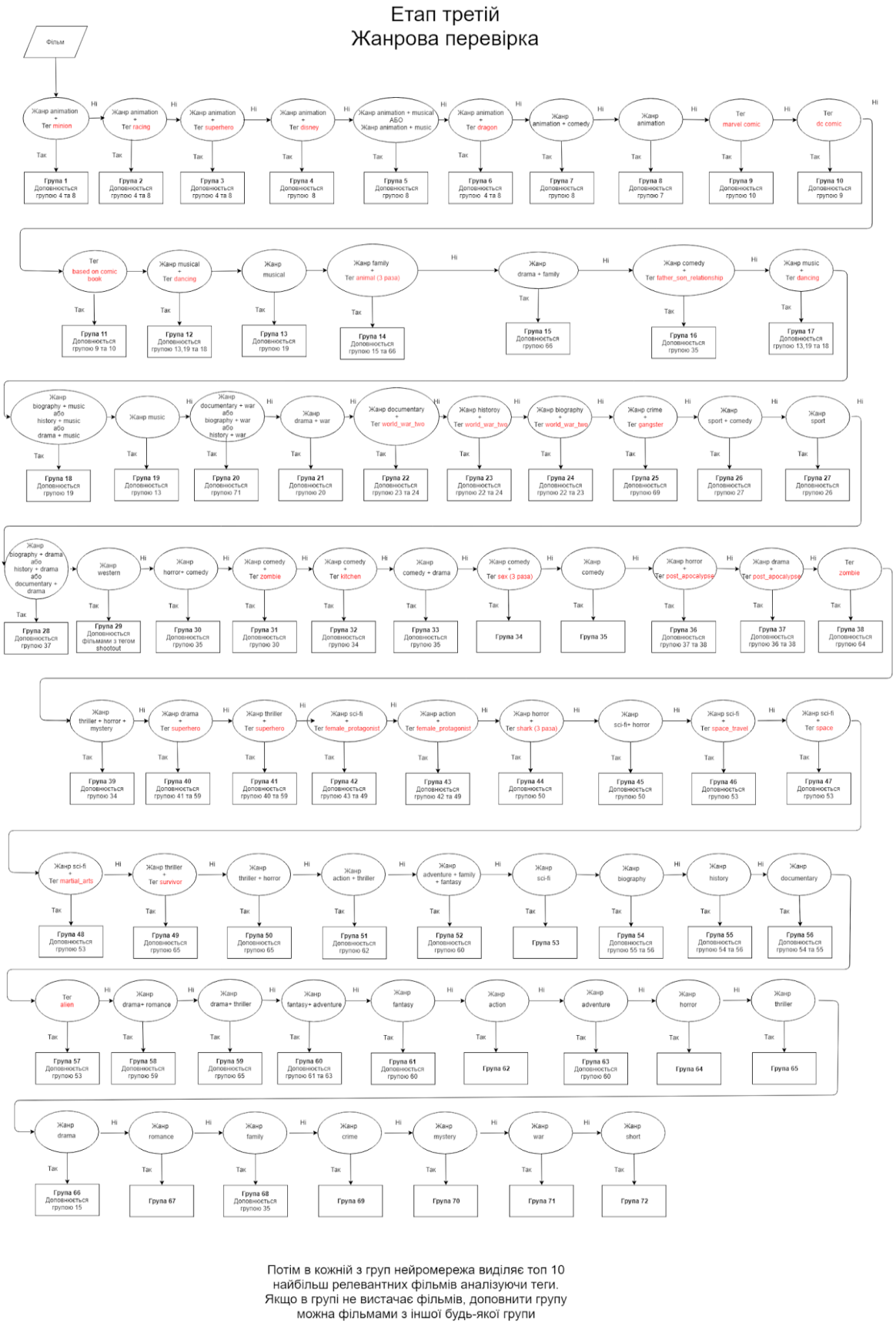


Рис. 3.4. Знаходження груп фільмів за допомогою жанрів і ключових слів

3.3. Формування списку схожих фільмів

Четвертим етапом є пошук схожих фільмів. На цьому етапі кожному фільму ставиться в пару 10 схожих на нього фільмів, відсортованих за оцінкою схожості. Це відбувається в кілька етапів. (рис. 3.5.).

Рис. 3.5. Пошук схожих фільмів

Спершу обирається фільм, для якого треба знайти аналоги. Потім визначається в якій групі знаходиться обраний фільм за допомогою сортувань за країною-виробником, віковими обмеженнями. Якщо фільмів в групі менше ніж 10 - група доповнюється іншою суміжною групою яка найбільш схожа за думкою кіно експерта. Далі використовується знайдена матриця схожості фільмів за допомогою ембедінгової моделі, яка навчається методом позитивно - негативних пар (3.2).

(3.2)

```
def generate_batch(pairs, pairs_set, films, tags, n_positive = 50, negative_ratio = 1.0):
    """Generate batches of samples for training"""
    batch_size = n_positive * (1 + negative_ratio)
```

```
batch = np.zeros((batch_size, 3))

neg_label = -1

# This creates a generator
while True:
    # randomly choose positive examples
    for idx, (film_id, tag_id) in enumerate(random.sample(pairs, n_positive)):
        batch[idx, :] = (film_id, tag_id, 1)

    # Increment idx by 1
    idx += 1
    # Add negative examples until reach batch size
    while idx < batch_size:

        # random selection
        random_film = random.randrange(len(films))
        random_tag = random.randrange(len(tags))

        # Check to make sure this is not a positive example
        if (random_film, random_tag) not in pairs_set:

            # Add to batch and increment index
            batch[idx, :] = (random_film, random_tag, neg_label)
            idx += 1

    # Make sure to shuffle order
    np.random.shuffle(batch)
    yield {'film': batch[:, 0], 'tag': batch[:, 1]}, batch[:, 2]
```

Для навчання моделі, що генерує перший ембедінг, створюються пари фільм - жанр, який притаманний цьому фільму та фільм - жанр, який не притаманний цьому фільму, те ж саме відбувається і з ключовими словами. Для генерування іншого ембедінгу використовуються пари жанр - фільм, який відповідає цьому жанру і навпаки, те ж саме і для ключових слів. Добуток цих ембедінгів дорівнює матриці ймовірностей, яка заповнена значеннями від 0 до 100.

Нейронна мережа за допомогою алгоритму оборотного поширення на основі градієнтного спуску мінімізує функцію помилки для знаходження найбільш точних значень схожості (3.3).

(3.3)

```
def train_model(pairs, pairs_set, tag_index, film_index, tags, films, epoch, batch_size):

    model = film_embedding_model(film_index, tag_index)


    gen = generate_batch(pairs, pairs_set, films, tags, batch_size, negative_ratio = 2)

    model.fit_generator(gen, epochs = epoch,
                        steps_per_epoch = len(pairs) // batch_size,
                        verbose = 2)

    return model
```

Після того як для обраного фільму ми отримали фільми, які відсортовані за оцінкою схожості (3.4), відбувається пост-обробка результатів за допомогою інформації про режисерів, акторів та назви фільму.

(3.4)

```
def get_similar_film_by_nn(name, model, film_index, index_film):

    film_layer = model.get_layer('film_embedding')
    weights = film_layer.get_weights()[0]
    weights = weights / np.linalg.norm(weights, axis = 1).reshape((-1, 1))

    index = film_index

    # Check to make sure `name` is in index
    try:
      # Calculate dot product between book and all others
      dists = np.dot(weights, weights[index[name]])
    except KeyError:
        print(f'{name} Not Found.')
        return

    # Sort distance indexes from smallest to largest
    sorted_dists = np.argsort(dists)

    # Take the last n sorted distances
    closest = sorted_dists[::-1][1:]
    #   closest = sorted_dists[:]

    return closest, dists[sorted_dists[::-1][1:]]
```

Якщо у фільмів однаковий режисер який входить у список відібраних режисерів кіно експертом тоді обидва фільми вважаються схожими на 98%, тому що відомі режисери зазвичай знімають кіно яке так чи інакше схоже за мотивами, навіть якщо це різні жанри. Також перевіряється кількість схожих акторів у фільмі, так як велика кількість клієнтів полюбляють певних акторів, і мають більші зацікавленості не в самому фільмі, а в грі їх улюблених акторів.

Далі проводиться аналіз на назву фільму. Якщо у назві фільмів різниця лише в одній цифрі - то це приквел або сиквел, такі фільми вважаються схожими на 98.5%. Знаходження схожих фільмів також реалізовано як мікросервіс. (рис. 3.6.).

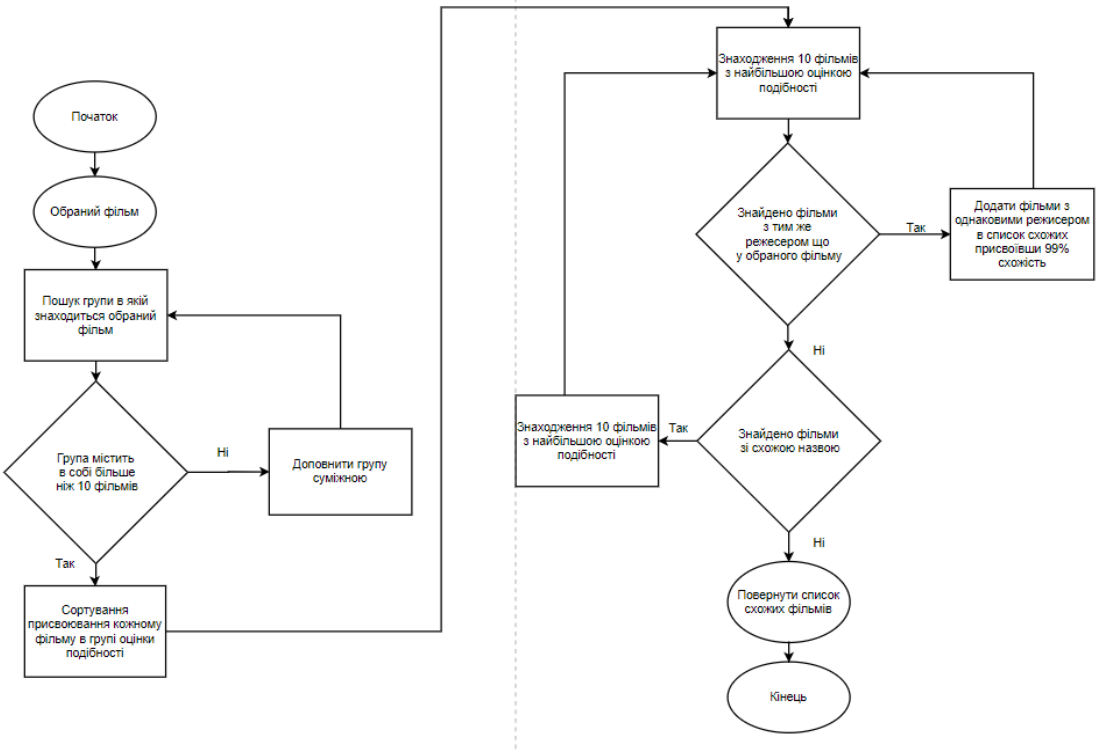


Рисунок 3.6. Реалізація системи знаходження схожих фільмів

Наступна частина системи – знаходження коефіцієнту релевантності фільму до клієнта через систему схожих фільмів (рис. 3.7.).

Рис. 3.7. Знаходження коефіцієнту релевантності через систему схожих фільмів

Щоб створити рекомендації, список фільмів в прокаті потрапляє до мікросервісу, де для них знаходяться схожі фільми. Далі для кожного клієнта знаходяться коефіцієнти релевантності до всіх фільмів зі списку.

3.4. Знаходження коефіцієнтів релевантності фільмів відносно клієнта

Для знаходження коефіцієнта релевантності між фільмом та клієнтом аналізується історія придбання білетів клієнтом, і якщо клієнт раніше вже купував білети на фільми, які є схожими до нового фільму, то знаходиться середнє значення оцінок схожості, що й буде коефіцієнтом релевантності.

За списком внутрішніх ID клієнтів завантажуються статистичні дані з Бази продажів. Інформація, що отримується, виглядає наступним чином:

Таблиця 3.2. Інформація з бази продажів

Поле	Значення	Тип поля
Client_ID	Внутрішній код клієнта	int
Film_ID	Внутрішній код фільму	int
Date	Дата сеансу	datetime
Price	Ціна білету	float
Amount	Кількість куплених білетів	int
Session_time	Проведений час на сторінці	timedelta

Якщо ж клієнт не купував білетів на жоден із списку схожих фільмів, то для такого клієнта знаходиться вектор жанрової зацікавленості шляхом аналізу всіх фільмів, які він відвідав раніше. (рис. 3.8.).

Рис. 3.8. Знаходження коефіцієнту релевантності через жанрову зацікавленість

Наступним кроком є знаходження косинусної відстані між вектором жанрової зацікавленості клієнта та жанрового вектору фільму. Ця відстань і буде коефіцієнтом релевантності.

Фільми, коефіцієнти релевантності яких були відібрані за допомогою коефіцієнтів схожості фільмів завжди будуть передувати фільмам які були відібрані за допомогою жанрової зацікавленості.

Отримавши всі коефіцієнти релевантності, фільми можуть бути проранжовані для створення особистих рекомендаційних листів для кожного клієнта.

Якщо ж клієнт в мобільному додатку поставив вподобання для якогось із фільмів, фільм автоматично буде першим у списку.

За списком внутрішніх ID клієнтів завантажуються статистичні дані з Бази інформації про клієнта. Інформація, що отримується, виглядає наступним чином:

Таблиця 3.3. Інформація з «Бази інформації про клієнта»

Поле	Значення
ClientID	Внутрішній код клієнта
Film_ID	Внутрішній код фільму
Date	Дата
Price	Ціна білету
Session_time	Проведений час на сторінці
Is_liked	Чи поставив клієнт вподобання

Далі таблиця проранжованих фільмів для кожного клієнта записується у внутрішню базу даних кінотеатру.

Таблиця використовується для створення розсилок на електронну пошту або ж для ранжування фільмів у мобільному додатку для кожного клієнта окремо. (рис. 3.9., рис. 3.10.).

Рис. 3.9. Приклад ранжування фільмів в мобільному додатку

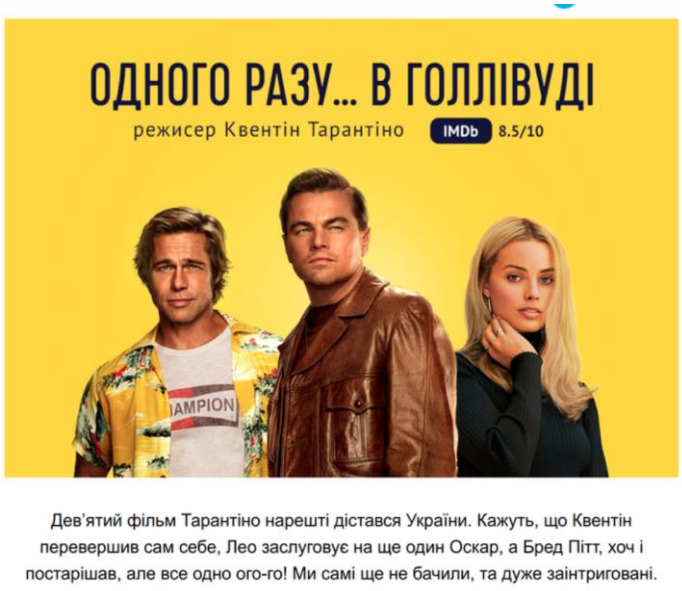


Рис. 3.10. Приклад розсилки на електронну пошту

На цьому розробка алгоритму знаходження коефіцієнтів релевантності (Додаток А рисунок 1) завершується і починається налаштування моделі знаходження матриці схожості фільмів. Оскільки метрики оцінювання схожості фільмів немає, а задача схожа на задачу кластеризації, було прийнято рішення обрати ембедінги невеликого розміру та тренувати модель поки внутрішня помилка оптимізації не буде покращена.

Задача знаходження схожих фільмів дуже подібна до задачі кластеризації об’єктів, але через те, що даних недостатньо щоб описати фільм для використання звичайних методів кластеризації, було вирішено використовувати підходи для обробки природної мови. Це дало змогу кластеризувати фільми з достатньою точністю щоб результат задовольнив кіно експертів.

ВИСНОВКИ ДО РОЗДІЛУ 3

Таким чином була створена система, яка дозволяє виконувати збір, оновлення та обробку даних з найменшим навантаженням на внутрішню базу даних кінотеатру та мінімальною кількістю запитів на головний сервер сховища даних.

Основною задачею системи є ранжування нових фільмів за допомогою пошуку схожих фільмів, що реалізований за допомогою навчання ембедінгової нейронної мережі, яка дозволяє отримати матрицю подібності фільмів на основі описових даних які включають в себе ключові слова та жанри фільмів.

Описові дані використовуються для створення навчальної вибірки яка складається з негативно-позитивних пар, за допомогою яких навчаються ембедінги нейронної мережі.

Матриця схожості фільмів використовується для сортування фільмів в групах фільмів, до яких потрапляють фільми за правилами, які встановив кіно експерт. Фільми з різними віковими лімітами мають бути в різних групах. Також відбувається розділення груп фільмів по країні-виробнику.

На першому місці в списку схожих фільмів також мають стояти фільми сиквели приквели, або ж фільми зняті одним і тим же режисером зі списку який був попередньо сформований кіно експертом.

Наступним кроком у формуванні рекомендацій є розрахунок коефіцієнтів релевантності які знаходяться за допомогою знайдених схожих фільмів до фільмів в прокаті. Їх пошук виконується за допомогою усереднення коефіцієнтів схожості між схожими фільмами які були знайдені мікросервісом пошуку схожих фільмів та фільмами на які клієнт вже купував білети.

Якщо ж серед схожих фільмів клієнт не купував нічого - для нього вираховується вектор жанрової зацікавленості базуючись на його попередніх переглядах. Коефіцієнти релевантності використовуються для сортування

фільмів в порядку, який буде найбільш релевантним для кожного клієнта особисто.

Також виконується пост-обробка списку проранжованих фільмів, де на перше місце ставляться фільми, які були відмічені бажаними в мобільному додатку. Відсортовані фільми можуть бути використані для створення повідомлень на електронну пошту про те, що в прокаті з'явився фільм, який точно зацікавить клієнта.

РОЗДІЛ 4
ОПИС ТЕСТУВАННЯ СИСТЕМИ

Тестування спроектованої системи проводилось в три етапи під час яких перевіряється робота всіх модулів.

4.1. Перший етап тестування

Спершу була проведена перевірка функціоналу збору та оновлення даних. Було проведено перевірку якості отриманих даних (рис. 4.1.) із зовнішнього джерела OMDB.

	movieId	movieName	movieNameUkr	movieUid	now	imdb	country	director	trailerUrll
0	1	By the Sea	Біля моря	00000000000000000000000000000164	0.0	tt3707106	France	Angelina Jolie	https://imdb-video.media-imdb.com/vi497725977/...
1	4	The Hateful Eight	Мерзенна вісімка	00000000000000000000000000000166	0.0	tt3460252	USA	Quentin Tarantino	https://imdb-video.media-imdb.com/vi2079306521...
2	9	The Secret Life of Pets	Секрети домашніх тварин	00000000000000000000000000000173	0.0	tt2709768	Japan	Chris Renaud	https://imdb-video.media-imdb.com/vi2386869273...
3	10	Alice Through the Looking Glass	Аліса в задзеркаллі	00000000000000000000000000000174	0.0	tt2567026	USA	James Bobin	https://imdb-video.media-imdb.com/vi2708190489...
4	11	Creed	Крід: Спадок Роккі Бальбоа	00000000000000000000000000000175	0.0	tt3076658	USA	Ryan Coogler	https://imdb-video.media-imdb.com/vi3155604249...

Рис. 4.1. Приклад отриманих даних

Перевірка показала наявність усіх потрібних полів даних. Та підтвердження відповідності актуальної інформацію з основним джерелом даних - IMDB.

Тестування виконувалося автоматично за допомогою створеного скраперу. Модуль збору даних для заповнення всієї бази починає роботу о 3 годині ранку і працює протягом 53-х хвилин. Для оновлення всієї бази потрібно 10 хвилин. Якість даних була перевірена на наявність пустих полів, що дорівнює 3%. Результатом перевірки системи є підтвердження коректності роботи модуля.

4.2. Другий етап тестування

Наступним етапом було тестування якості алгоритму підбору схожих фільмів. Перевірка виконувалася експертами зі сторони кінцевого користувача. Результат тестування показав достатньо велику коректність роботи підбору фільмів.

Алгоритм користування цим модулем простий:

1) Перейти на сторінку з вибором фільмів, які потрібно проаналізувати
2) Вибрати фільм із зони «Випадково обрані» (рис. 4.2.).

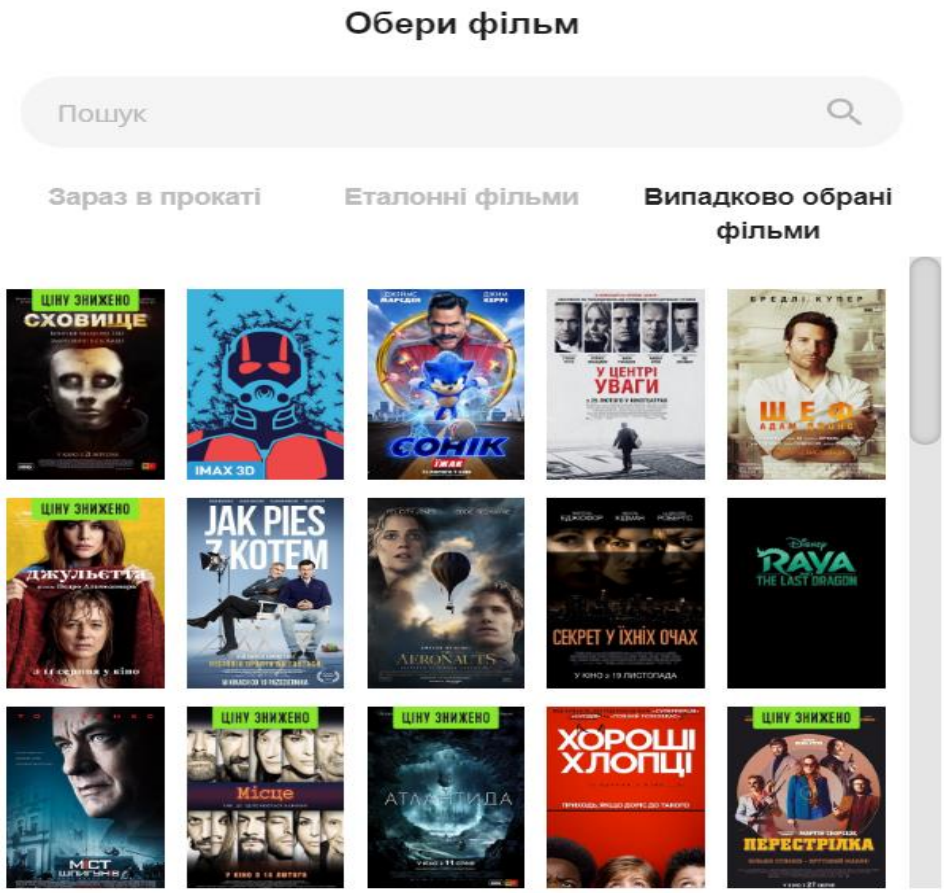


Рис. 4.2. Вікно вибору фільму для аналізу

Після цього з’явиться сторінка, де посередині буде постер обраного фільму. Зправа від постера в описовому вікні буде інформація про жанри, країну-виробника, режисера та ключові слова. Зліва від постера знаходиться

трейлер фільму, який можна швидко переглянути, щоб зрозуміти посил фільму (рис. 4.3.).

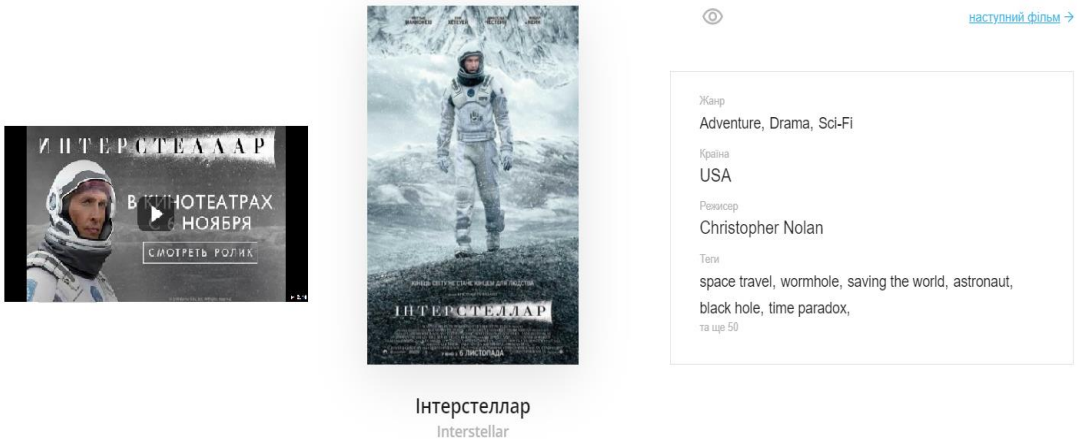


Рис. 4.3. Інтерфейс опису обраного фільму

Внизу, під постером, будуть знаходитися фільми, які йдуть в порядку спадання зліва направо щодо оцінки подібності. Для того щоб дізнатися який саме відсоток подібності потрібно натиснути на картинку ока, яка розташована вище описового вікна. Також реалізована можливість переглянути за якими ознаками фільми схожі - для цього потрібно навести курсор на будь-який фільм зі списку схожих. Внизу, під постером схожого фільму, відобразиться текст, де буде написано вікові обмеження фільму, групу з жанрів та ключових слів, до якої належить фільм, та суміжну групу, якщо фільм із суміжної групи (рис 4.4.).

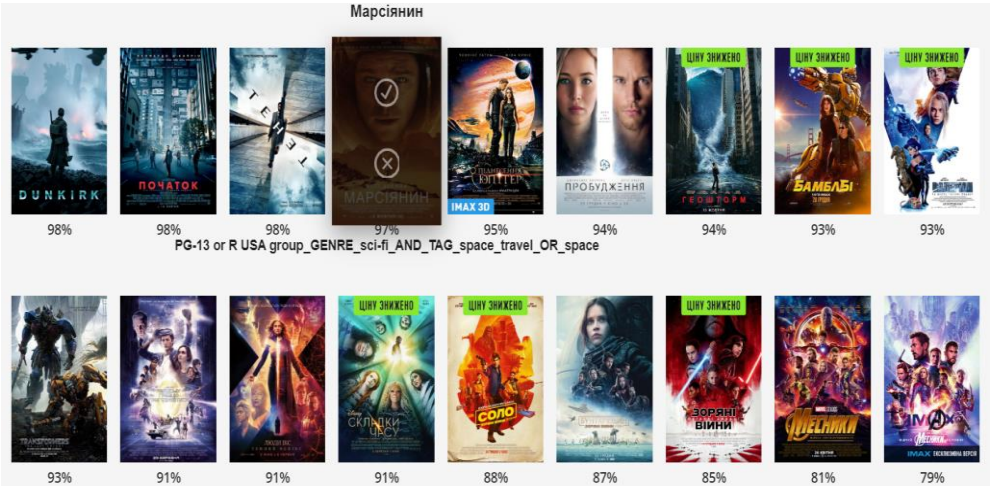


Рис. 4.4. Приклад підбору схожих фільмів

Якщо ж фільми мають однакового режисера - він буде перший в списку схожих (рис 4.5).

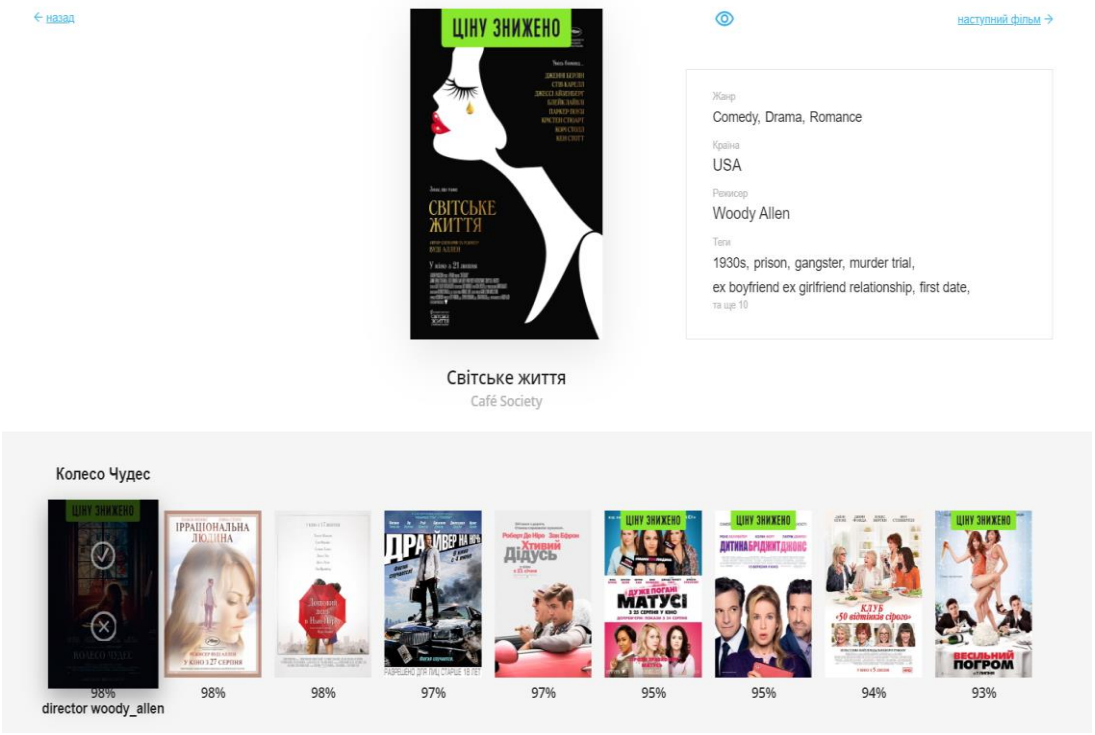


Рис. 4.5. Тестування положення фільму з однаковим режисером

Фільми сиквели або приквели теж мають бути на першому місці в списку схожих (рис 4.6).

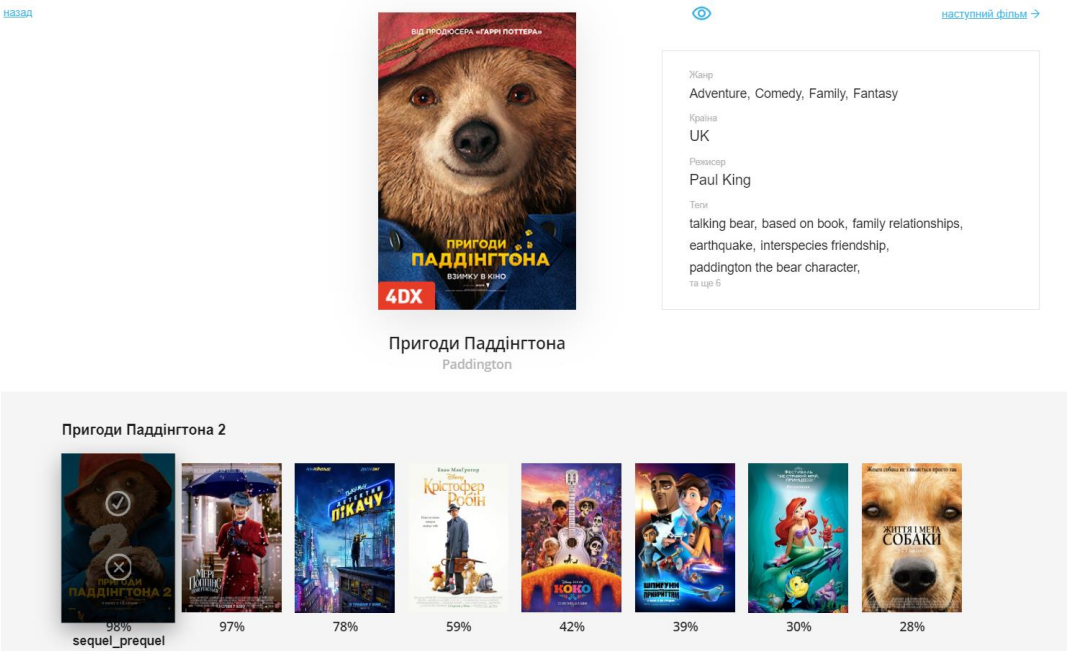


Рис. 4.6. Тестування положення фільму приквелу або сиквелу

До мультфільма мають знаходитися лише мультфільми (рис. 4.7) через те, що вікові обмеження не дозволяють до мультиків радити фільми які можуть дивитися тільки дорослі.

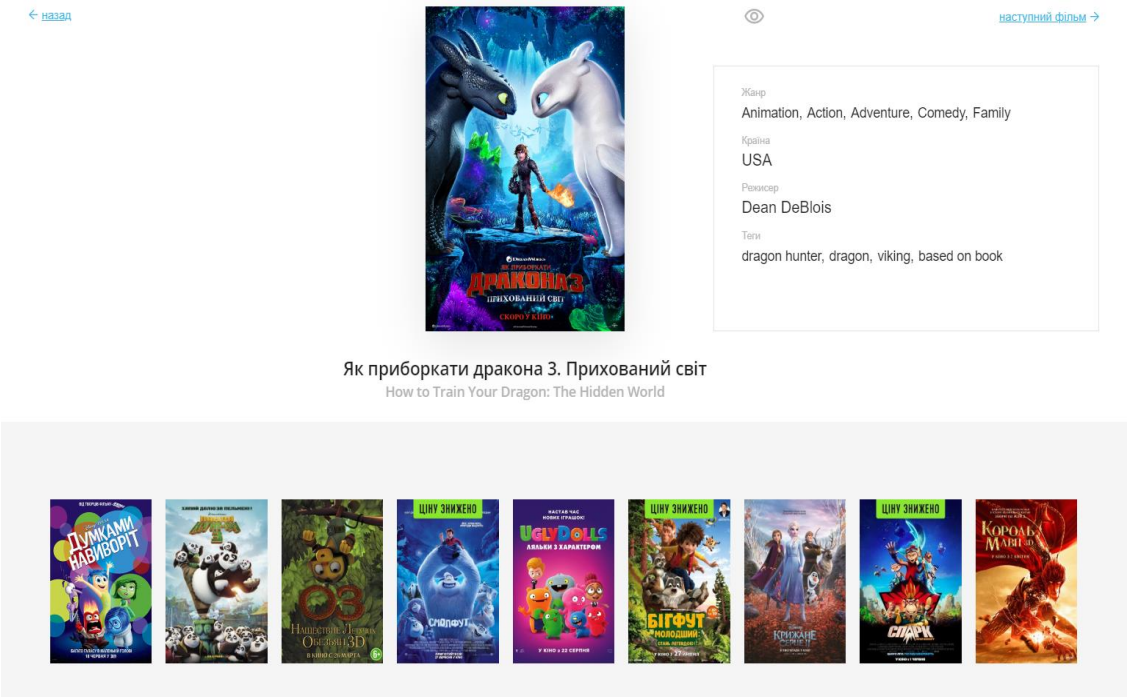


Рис. 4.7. Тестування положення фільму приквелу або сиквелу

Через те, що задача кластеризації не має метрик оцінювання з математичним обґрунтуванням, тестування модуля також проводилося кіно-експертом, який підтвердив коректність роботи модуля.

4.3. Третій етап тестування

Також потрібно було провести тестування коректності роботи алгоритму створення рекомендаційних списків для кожного клієнта. Так як оцінка ранжування не має метрики яка обґрунтована математично, було вирішено протестувати декілька тест сценаріїв на коректність роботи алгоритму.

Спочатку було перевірено що клієнту, який в мобільному додатку

поставив фільму вподобання, цей фільм буде першим в рекомендаційному листі.

Далі було перевірено що фільми які рекомендуються через модуль схожих фільмів працює правильно. Для цього було взято тестового клієнта, створено тест сценарій, який мав очікуваний результат. Після обробки цього клієнта рекомендаційною системою результати співпали.

Таким же чином було перевірено фільми які рекомендуються через вектор зацікавленості.

Також було проведено тестування системи на групі з 20 реальних людей, які працюють в кінотеатрі та мають історію придбаних білетів. Для них було створено їх особисті рекомендаційні листи.

Результатом перевірки є 17 людей, які задоволені результатом рекомендаційного листа, 2-є людей не змогли надати точної відповіді і 1 людина не задоволена підбором.

ВИСНОВКИ ДО РОЗДІЛУ 4

Тестування створеного програмного продукту було проведено в три етапи під час яких було перевірено роботу всіх модулів системи яка складається з модуля збору даних, модуля пошуку схожих фільмів та модуля створення рекомендацій.

За результатами першого етапу було перевірено коректність роботи модуля збору та оновлення даних.

Тестування підтвердило якісну роботу збору інформації, актуальність інформації та наявність потрібних полів даних. Також було перевірено ступінь нагрузки на основну систему кінотеатру через запити які виконує мікросервіс. Тестування показало низький ступінь навантаженості .

За результатами тестування другого етапу експертна оцінка моделі підбору схожих фільмів була оцінена як хороша.

Модуль знаходження схожих фільмів також був протестований на робітниках кінотеатру.

Робота модуля сподобалася 30-ти робітникам з 35 наявних тестувальників і було вирішено винести роботу модуля як окремий продукт, за допомогою якого можна знаходити схожі фільми до тих, які дуже сподобалися клієнту.

Під час тестування було перевірено функції знаходження схожих фільмів за спільним режисером або акторами. Фільми сиквели та приквели були також завжди на першому місці в списку схожих фільмів.

За результатами третього етапу було підтверджено, що алгоритм влучно створює рекомендаційні листи. Перевірено, що завжди для клієнтів які поставили вподобання фільму в мобільному додатку, цей фільм на першому місці.

Також система була запущена для двох тисяч бета-клієнтів з різних куточків країни, які були вибрані випадковим чином. Бета-тестування показало збільшення відгуку переглядів від тестової групи клієнтів. Загалом було відстежено вплив рекомендацій через різні шляхи комунікації.

Збільшення куплених білетів відбулося по переходу на білінгову систему через рекламу на пошті на 16%, через вайбер на 11%, через мобільний додаток на 23% порівняно з тими клієнтами до яких рекомендаційна система не була застосована.

ВИСНОВКИ

Було спроектовано та розроблено автоматизовану рекомендаційну систему фільмів для кінотеатрів на основі статистичних даних.

Було досліджено предметну область формування рекомендаційних листів для користувачів кінотеатрів. Розглянуто метод вирішення задачі створення рекомендацій, а саме - використання різних алгоритмів підбору схожих фільмів. Було обрано сучасний метод обробки текстових даних, який полягає у створенні позитивно - негативних пар для навчання моделі, яка використовує ембедінги для визначення схожості об’єктів.

Також ми проаналізували основні вимоги користувача до програмної реалізації системи.

Було проведено аналіз часткових готових аналогів програмних рекомендаційних систем для формування листів клієнту. Виявлено слабкі місця в аналогів програмних продуктів, а серед їхніх основних недоліків є те, що рекомендації створюються лише після того, як з фільмом вже провзаємодіяла якась частина клієнтів.

Була побудована нейронна мережа для автоматизованого формування рекомендаційних листів.

Було створено програмний продукт, що дав змогу збирати та оновлювати локальну базу даних фільмів, знаходити схожі фільми з минулих прокатів та по ним створити алгоритм, який знаходить коефіцієнти релевантності для кожного клієнта по кожному фільму.

Також ми провели тестування рекомендаційної системи в три етапи.

За результатами першого етапу було перевірено, що модуль збору та оновлення даних працює швидко, без перебоїв. Якість зібраних даних є високою.

За результатами тестування другого етапу експертна оцінка моделі підбору схожих фільмів була оцінена як робоча.

За результатами третього етапу було підтверджено, що алгоритм влучно створює рекомендаційні листи, але сама точність цифр відносна.

Отже, таким чином був розроблений програмний продукт, який повністю відповідає сучасним вимогам та є ефективним інструментом для створення рекомендаційних листів для кожного клієнта на основі статистичних даних, а також не має аналогів в Україні.

ПЕРЕЛІК ПОСИЛАНЬ

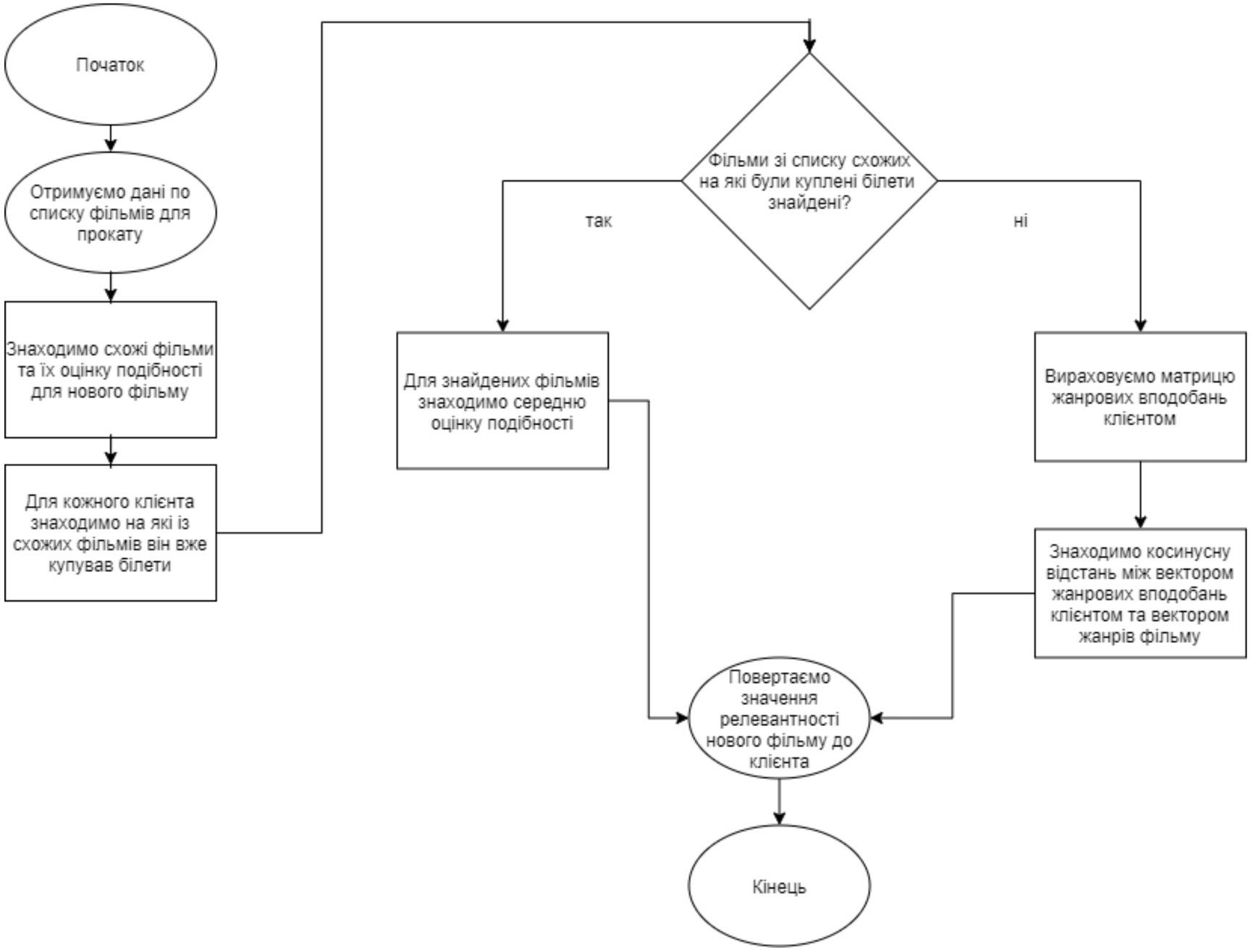
1. Rolt L. T. C., Allen J. S. The Steam engine of Thomas Newcomen. Landmark, Ashbourne, 1997. 44 p.
2. Неуймин Я. Г. Модели в науке и технике. История, теория, практика, Ленинград: Наука, Ленинградское отделение, 1984. 190 с.
3. Самарский А.А., Михайлов А.П. Математическое моделирование. М.: ФИЗМАТЛИТ, 2002. 320 с.
4. Бейли Н. Математика в биологии и медицине. – М.: Мир, 1970. 328 с.
5. Павловский Ю.Н. Имитационные модели и системы. ФАЗИС, ВЦ РАН. 2000. 144 с.
6. Попов Ю.П., Самарский А.А. Вычислительный эксперимент. М.: Знание, 1983. 64 с.
7. Плохотников К.Э. Математическое моделирование и вычислительный эксперимент. Едиториал УРСС, 2003. 280 с.
8. Петров А.А. Экономика. Модели. Вычислительный эксперимент. М.: Наука, 1996. 251 с.
9. Чучуева И. А. Модель прогнозирования временных рядов по выборке максимального подобия. Московский государственный технический университет им. Н. Э. Баумана. Москва, 2012
10. Нейт Сильвер. Сигнал и Шум. Почему одни прогнозы сбываются, а другие — нет // Азбука-Аттикус, КоЛибри, 2015. 190 с.
11. Обзор методов прогнозирования. URL:https://ivan-shamaev.ru/overview-forecast-methods/ https://ivan-shamaev.ru/overview-forecast-methods/ (дата звернення: 04.05.2020).
12. Mathematical model. URL: https://www.sciencedaily.com/terms/mathematical_model.htm (дата звернення 05.05.2020)
13. Dr. Anasse Bari, Mohamed Chaouchi, Tommy Jung. Predictive Analytics For Dummies // For Dummies; 2 edition, 2016. 86 с.

14. Cameron Davidson-Pilon. Bayesian Methods for Hackers: Probabilistic Programming and Bayesian Inference // Addison-Wesley Data & Analytics, 2015. 22 p.
15. John K. Kruschke. Doing Bayesian Data Analysis, Second Edition: A Tutorial with R, JAGS, and Stan // Academic Press / Elsevier, 2015. 193 p.
16. Евгений Бурнаев, Федор Губарев, Сергей Морозов, Александр Прохоров, Дмитрий Хоминич. Многодисциплинарная оптимизация, анализ данных и автоматизация инженерных расчетов с помощью программного комплекса pSeven // CAD/CAM/CAE Observer #4 (88), 2014. 443 с.
17. Max Kuhn, Kjell Johnson. Applied Predictive Modeling // Springer, 2013. 293 с.
18. Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction // A Bradford Book; 1St Edition edition (March 1, 1998). 293 с.
19. Бахвалов Н. С. Численные методы. 3-е изд. — М., 2003. 113 с.
20. U. Rüde, K. Willcox, L. C. McInnes et al. Research and Education in Computational Science and Engineering. 197 p.
21. Gilbert Strang. Computational Science and Engineering. // Wellesley-Cambridge Press, 2007. 113 p.
22. Christopher Bishop. Pattern Recognition and Machine Learning // Springer, 2006
23. Mikhail Belyaev, Evgeny Burnaev, Ermek Kapushev, Maxim Panov, Pavel Prikhodko, Dmitry Vetrov, Dmitry Yarotsky. GTApprox: Surrogate modeling for industrial design // Advances in Engineering Software 102 (2016) 29–39. 193 p.
24. Yann LeCun, Yoshua Bengio, Geoffrey Hinton. Deep learning // Nature 521, 436–444 (28 May 2015). 233 p.
25. Mohamad H.Hassoun. Fundamentals of Artificial Neural Networks. MIT Press, Cambridge, Massachusetts, 1995.. 93 p.

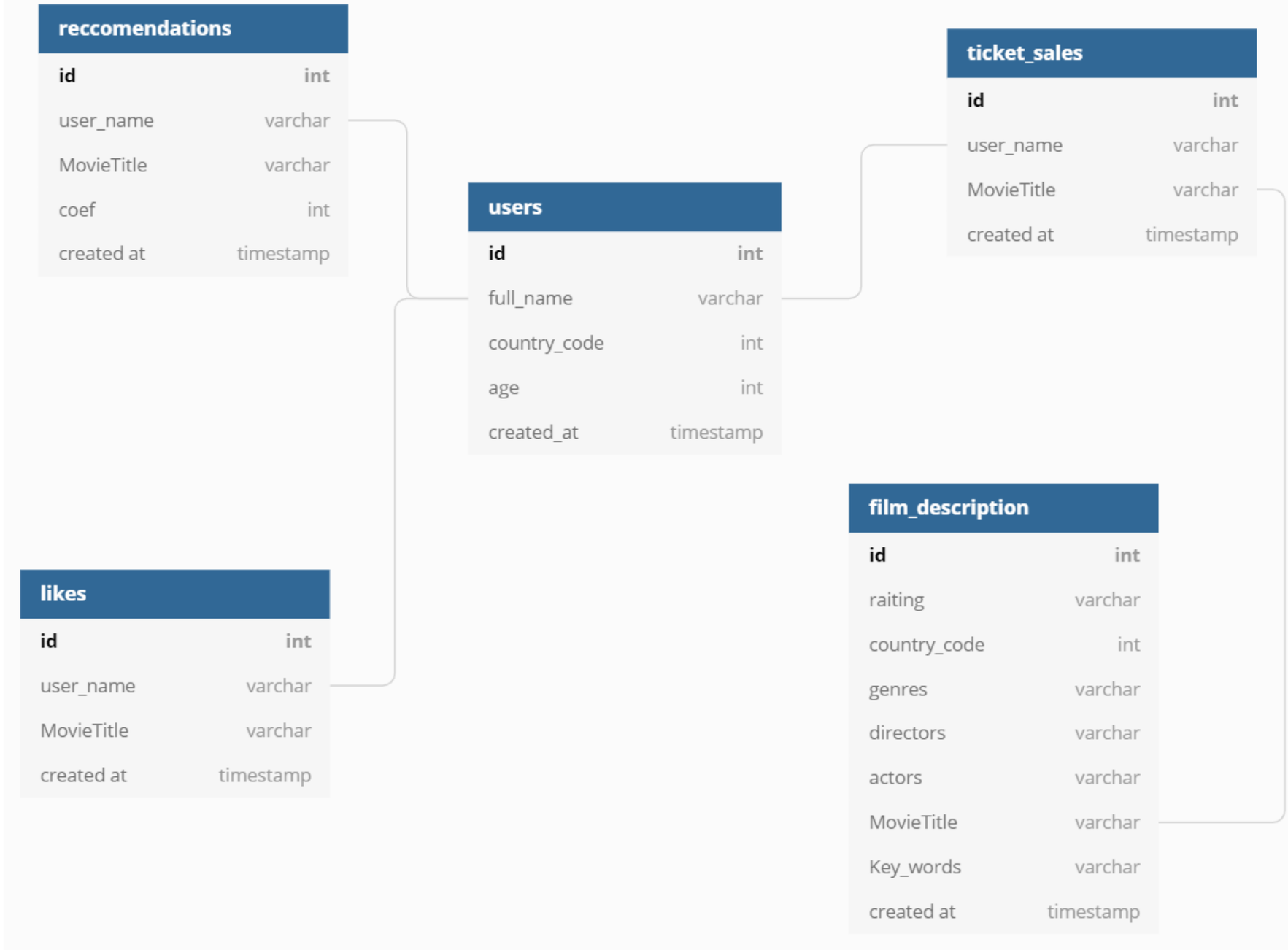
26. S.Haykin. Neural Networks and Learning Machines. 3rd Edition. Pearson, 2018. 33 p.
27. C.C.Aggarwal. Neural Networks and Deep Learning. A Textbook. Springer International Publishing. 223 p.
28. В.В.Круглов, М.И.Дли, Р.Ю.Голунов. Нечеткая логика и искусственные нейронные сети. Физматлит, 2001. 113 с.
29. Ахо А., Хопкрофт Дж., Ульман Дж. построение и анализ вычислительных алгоритмов. М.:Мир.-1979. – 420 с.
30. Березин А.С., Петренко С.А. Защита информации в открытых сетях // Корпоративные системы.- 2001.-№ 2.- С.65-69.
31. Березовский А.И., Задирака В.К., Шевчук Л.Б. О тестировании быстродействия алгоритмов и программ вычисления основных операций ассиметричной криптографии /Кибернетика и системный анализ №5, 1999. – С. 59-66.
32. Зайченко Ю. П. Исследование операций (задачник). Киев, 2003. 250 с.
33. Щербакова И. В. Математическое моделирование информационных систем центров ситуационного управления в интересах обеспечения безопасности: автореф. дис.канд. техн. наук. Воронеж: ВИ МВД России, 2009. 16 с.
34. Хаббард Дж. Автоматизированное проектирование баз данных. Дж. Хаббард М.: Мир, 1983. 295 с.
35. Шеннон Р. Имитационное моделирование систем. Искусство и наука: пер. с англ. М.: Мир, 1978. 418 с.
36. Фейгин Л. И. Задачи теории расписаний при нечетких длительностях операций . Л.И. Фейгин. Доклады АН СССР, 1983, т. 272, №4, с. 812–815.
37. Бокс Дж., Дженкинс Г. М. Анализ временных рядов, прогноз и управление. М.: Мир, 1974. 406 с.

38. Armstrong J.S. Forecasting for Marketing. Quantitative Methods in Marketing. London: International Thompson Business Press, 1999. P. 92 – 119.
39. Draper N., Smith H. Applied regression analysis. New York: Wiley, In press, 1981. 693 p.
40. Сидоров С. Г., Никологорская А. В. Анализ временных рядов как метод построения потребления электроэнергии . Вестник ИГЭУ. 2010, Вып. 3. С. 81–83.
41. Мерков А. Б. Распознавание образов: Введение в методы статистического обучения. М.: Едиториал УРСС, 2011. 254 с.
42. Тихонов Э. Е. Прогнозирование в условиях рынка. Невинномысск, 2006. 221 с.

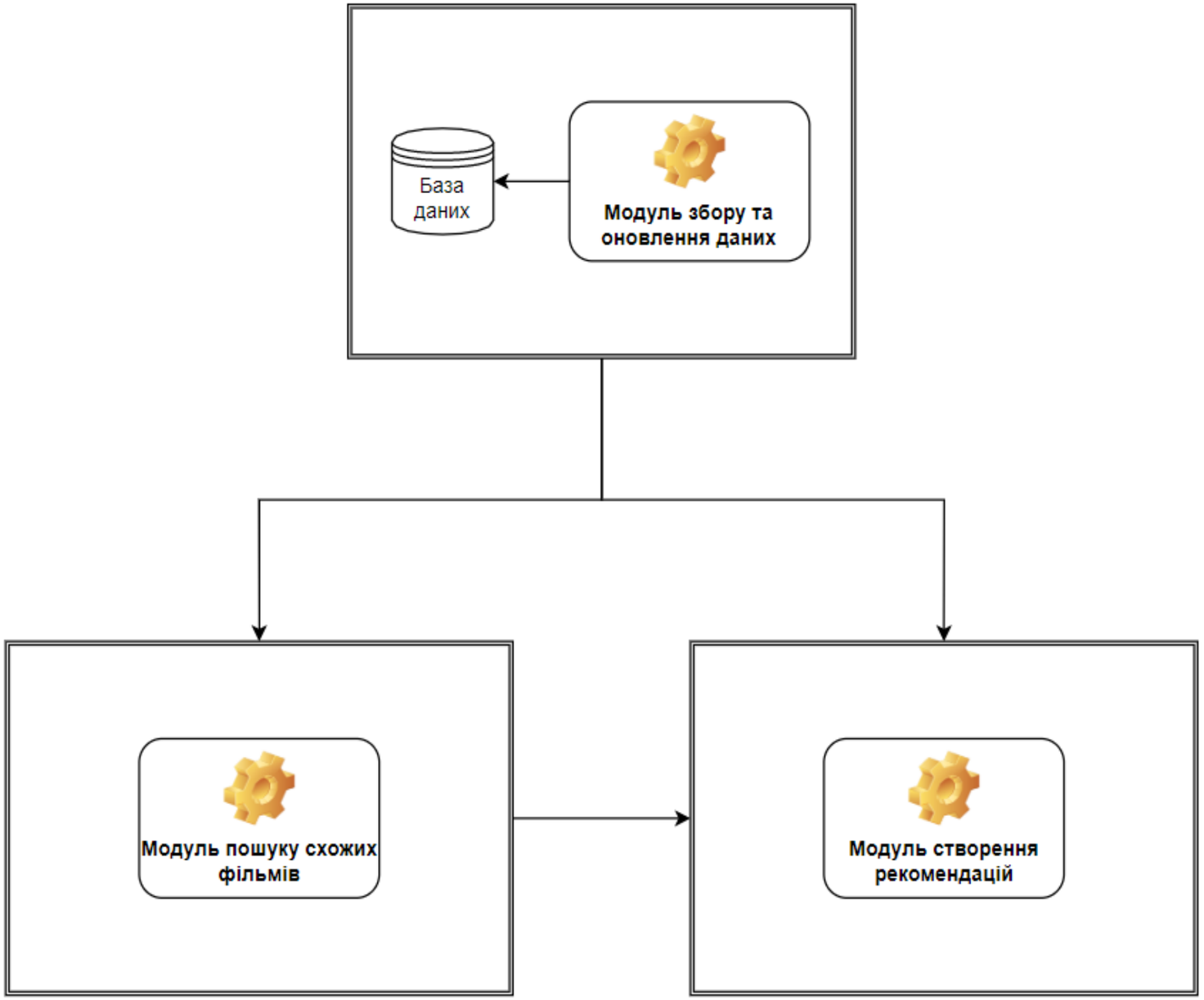
ДОДАТОК А



					ІАЛЦ.006305.004 Д1	
Зм.	Арк.	№ докум.	Підпис	Дата	Алгоритм створення рекомендацій фільмів Схема принципова	Літера / Маса / Масштаб
Розроб.		Братун А.Ю.				
Перевір.		Новотарський М.А.				Аркуш 1 / Аркушів 1
Т. контр.						НТУУ "КПІ" ФІОТ Група ІО-63
Н. контр.		Сімоменко В.І.				
Затв.						



					ІАЛЦ.006305.005 Д2		
Зм.	Арк.	№ докум.	Підпис	Дата	Структура сховища даних Схема функціональна	Літера	Маса / Масштаб
Розроб.		Братун А.Ю.					
Перевір.		Новотарський М.А.				Аркуш 1	Аркушів 1
Т. контр.						НТУУ "КПІ" ФІОТ Група ІО-63	
Н. контр.		Сімоменко В.П.					
Затв.							



					ІАЛЦ.006305.006 ДЗ	
Зм.	Арк.	№ докум.	Підпис	Дата	Взаємодія модулів в системі Схема структурна	Літера / Маса / Масштаб
Розроб.		Братун А.Ю.				
Перевір.		Новотарський М.А.				Аркуш 1 / Аркушів 1
Т. контр.						
Н. контр.		Сімоменко В.П.				НТУУ "КПІ" ФІОТ Група ІО-63
Затв.						

ДОДАТОК Б

```
# Seed value
# Apparently you may use different seed values at each stage
seed_value = 0
import pandas as pd

# 1. Set the `PYTHONHASHSEED` environment variable at a fixed value
import os
os.environ['PYTHONHASHSEED'] = str(seed_value)

# 2. Set the `python` built-in pseudo-random generator at a fixed value
import random
random.seed(seed_value)

# 3. Set the `numpy` pseudo-random generator at a fixed value
import numpy as np
np.random.seed(seed_value)

# 4. Set the `tensorflow` pseudo-random generator at a fixed value
import tensorflow as tf
tf.set_random_seed(seed_value)

# 5. Configure a new global `tensorflow` session
from keras import backend as K
session_conf = tf.ConfigProto(intra_op_parallelism_threads=1, inter_op_parallelism_threads=1)
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)

import pymssql
from fuzzywuzzy import fuzz
import requests
from stemming.porter2 import stem
import pymysql
from pymysql.cursors import DictCursor
from dotenv import load_dotenv
from pathlib import Path  # python3 only
from keras.layers import Input, Embedding, Dot, Reshape
from keras.models import Model
from keras.models import load_model
from tqdm import tqdm


np.set_printoptions(suppress=True)


# import env variables
env_path = Path('.') / '.env'
load_dotenv(dotenv_path=env_path)



epoch_amount = 20

films_to_show = 100
tag_power = 14



# Function for prepare data for Neural Net
def make_pairs_for_train_NN(df):
```

```
flat_list_all_tags = [item for sublist in df.bag_of_words_for_train_nn.apply(lambda x: x.split(' ')) for item in sublist]

# Delete tags which happened once
st_wrds.extend(pd.Series(flat_list_all_tags).value_counts()[pd.Series(flat_list_all_tags).value_counts() == 1].index.tolist())
df.bag_of_words_for_train_nn = df.bag_of_words_for_train_nn.apply(lambda x: [i for i in x.split(' ') if i not in st_wrds])

film_index = {film: idx for idx, film in enumerate(df.long.values)}
index_film = {idx: film for film, idx in film_index.items()}

list_of_lists = df.bag_of_words_for_train_nn.values
flat_list = [item for sublist in list_of_lists for item in sublist]
tags = pd.Series(flat_list).value_counts().to_dict()
tag_index = {tag: idx for idx, tag in enumerate(tags)}

pairs = []
films = df.long.values

# Iterate through each film
for film in films:
# Iterate through the tags in the film
pairs.extend((film_index[film], tag_index[tag.lower()]) for tag in df[df.long ==
film].bag_of_words_for_train_nn.values[0] if tag.lower() in tags)

pairs_set = set(pairs)


return pairs, pairs_set, tag_index, film_index, index_film, tags, films

def generate_batch(pairs, pairs_set, films, tags, n_positive = 50, negative_ratio = 1.0):
"""Generate batches of samples for training"""
batch_size = n_positive * (1 + negative_ratio)
batch = np.zeros((batch_size, 3))


neg_label = -1

# This creates a generator
while True:
# randomly choose positive examples
for idx, (film_id, tag_id) in enumerate(random.sample(pairs, n_positive)):
batch[idx, :] = (film_id, tag_id, 1)

# Increment idx by 1
idx += 1

# Add negative examples until reach batch size
while idx < batch_size:

# random selection
random_film = random.randrange(len(films))
random_tag = random.randrange(len(tags))

# Check to make sure this is not a positive example
if (random_film, random_tag) not in pairs_set:

# Add to batch and increment index
batch[idx, :] = (random_film, random_tag, neg_label)
idx += 1

# Make sure to shuffle order
```

```
        np.random.shuffle(batch)
        yield {'film': batch[:, 0], 'tag': batch[:, 1]}, batch[:, 2]

def film_embedding_model(film_index, tag_index, embedding_size = 50):
    """Model to embed filns and tags using the functional API"""

    # Both inputs are 1-dimensional
    film = Input(name = 'film', shape = [1])
    tag = Input(name = 'tag', shape = [1])


    # Embedding the film (shape will be (None, 1, 50))
    film_embedding = Embedding(name = 'film_embedding',
                               input_dim = len(film_index),
                               output_dim = embedding_size)(film)

    # Embedding the tag (shape will be (None, 1, 50))
    tag_embedding = Embedding(name = 'tag_embedding',
                              input_dim = len(tag_index),
                              output_dim = embedding_size)(tag)

    # Merge the layers with a dot product along the second axis (shape will be (None, 1, 1))
    merged = Dot(name = 'dot_product', normalize = True, axes = 2)([film_embedding, tag_embedding])

    # Reshape to be a single number (shape will be (None, 1))
    merged = Reshape(target_shape = [1])(merged)


    model = Model(inputs = [film, tag], outputs = merged)
    model.compile(optimizer = 'Adam', loss = 'mse')

    return model

def get_similar_film_by_nn(name, model, film_index, index_film):

    film_layer = model.get_layer('film_embedding')
    weights = film_layer.get_weights()[0]
    weights = weights / np.linalg.norm(weights, axis = 1).reshape((-1, 1))

    index = film_index

    # Check to make sure `name` is in index
    try:
        # Calculate dot product between book and all others
        dists = np.dot(weights, weights[index[name]])
    except KeyError:
        print(f'{name} Not Found.')
        return

    # Sort distance indexes from smallest to largest
    sorted_dists = np.argsort(dists)

    # Take the last n sorted distances
    closest = sorted_dists[::-1][1:]
    #   closest = sorted_dists[:]

    return closest, dists[sorted_dists[::-1][1:]]


def train_model(pairs, pairs_set, tag_index, film_index, tags, films, epoch, batch_size):

    model = film_embedding_model(film_index, tag_index)
```

```
    gen = generate_batch(pairs, pairs_set, films, tags, batch_size, negative_ratio = 2)

    model.fit_generator(gen, epochs = epoch,
                        steps_per_epoch = len(pairs) // batch_size,
                        verbose = 2)

    return model

def get_chunk(df, LONG_ID, closest_films, film_index, original_or_kino_mama_dataset = None):
    '''Pass long id and get list of similar films'''

# If original_or_kino_mama_dataset == None, than chunk empty, else it alredy have similar kinomama_eng films
    # it uses in main function in bottom
    if original_or_kino_mama_dataset == None:
        chunk = []
    else:
        chunk = original_or_kino_mama_dataset

    # Film for which we find similar films
    film_for_reccomendations = df[df.long == LONG_ID]

    # dataframe, without film for which we find similar
    df = df[df.long != LONG_ID]


    # make film description
    current_rated = film_for_reccomendations.rated.values[0]
    current_country = film_for_reccomendations.country.values[0]
    current_keywords_by_space = film_for_reccomendations.keywords_by_space.values[0].split(' ')
    current_bag_of_words_for_train_nn = film_for_reccomendations.soup.values[0].split(' ')

    current_directors = film_for_reccomendations.director.values[0]
    current_ukrTitle = film_for_reccomendations.ukr.values[0]
    current_origTitle = film_for_reccomendations.originalTitle.values[0]
    current_actors = film_for_reccomendations.actors.values[0]

    current_status = film_for_reccomendations.status.values[0]

    ############################ sequel_prequel SORT ############################


def sequel_prequel_similaruty(dataset, current_director, current_actors, current_country, current_original_title,
                              current_ukr_title):
    if 'ukraine' in current_country or 'russia' in current_country:
        dataset = dataset[np.logical_or(dataset.country.str.contains('ukraine'), dataset.country.str.contains('russia'))]
    else:
        dataset = dataset[~np.logical_or(dataset.country.str.contains('ukraine'), dataset.country.str.contains('russia'))]

    if current_status == 'КіноМама':
        dataset = dataset[dataset.status.str.contains('КіноМама')]
    elif current_status == 'In English':
        dataset = dataset[dataset.status.str.contains('In English')]
    else:
        dataset = dataset[~np.logical_or(dataset.status.str.contains('КіноМама'), dataset.status.str.contains('In
                                                                                                               English'))]


    for index, row in dataset.iterrows():
        if ( set(row['director'].split(' ')).intersection(set(current_director.split(' '))) and (current_director != 'n/a' and
                                                                                                 row['director'] != 'n/a') ) or (len(set(row['actors'].split(' ')).intersection(set(current_actors.split(' ')))) >= 2):

            if 'ukraine' in current_country or 'russia' in current_country:
```

```
if ( (fuzz.partial_ratio(current_ukr_title, row['ukr']) > 75) and (current_ukr_title != 'n/a' and row['ukr']
!= 'n/a') ) \
    or (set(row['director'].split(' ')).intersection(set(current_directors.split(' '))) and
(len(set(row['actors'].split(' ')).intersection(set(current_actors.split(' ')))) >= 1) and ('genre_animation' not in
current_bag_of_words_for_train_nn) and (fuzz.partial_ratio(current_ukr_title, row['ukr']) > 50) and
(current_actors != 'n/a' and row['actors'] != 'n/a') ):
        print('here_start')
        print(current_ukrTitle)
        print(row['ukr'])
        print('here_end')
        if film_index[row['long']] not in chunk:
        chunk.append([film_index[row['long']], ' sequel_prequel'])

    else:
    if ( (fuzz.partial_ratio(current_original_title, row['originalTitle']) > 80) and (current_original_title != ''
and row['originalTitle'] != '') ) \
    or (set(row['director'].split(' ')).intersection(set(current_directors.split(' '))) and
(len(set(row['actors'].split(' ')).intersection(set(current_actors.split(' ')))) >= 1) and ('genre_animation' not in
current_bag_of_words_for_train_nn) and (fuzz.partial_ratio(current_original_title, row['originalTitle']) > 50) and
(current_actors != 'n/a' and row['actors'] != 'n/a') ):
        print('here_start')
        print(current_actors)
        print(current_ukrTitle)
        print(row['ukr'])
        print('here_end')
        if film_index[row['long']] not in chunk:
        chunk.append([film_index[row['long']], ' sequel_prequel'])


    dataset = df.copy()
    dataset.fillna('n/a', inplace=True)
sequel_prequel_similaruty(dataset, current_directors, current_actors, current_country, current_origTitle,
current_ukrTitle)


############################ DIRECTOR SORT ############################


# if film have similar directors than add in into chunk, and delete from dataframe
    for director in director_list:
    if director in current_directors:
    director_df = df[df.director.str.contains(director)]
    if not director_df.empty:
    reccomendation_by_director.extend(director_df.long.tolist())
reccomendation_by_director_index = [film_index[i] for i in reccomendation_by_director]
    for i in closest_films:
# films dont have to be present in chunk (maybe it can hit there from mama_orig or ?other? way)
    if i in reccomendation_by_director_index and i not in chunk:
    chunk.append([i, ' director ' + director])


############################ AGE SORT ############################


rated_pg_13 = df[df.rated.apply(lambda x: 'PG-13' in x)]

if 'G' in current_rated or 'PG' in current_rated:
```

```
rated_df = df[df.rated.apply(lambda x: 'G' in x or 'PG' in x)]
age_limit = 'G or PG'
else:
rated_df = df[~df.rated.apply(lambda x: 'G' in x or 'PG' in x)]
age_limit = 'PG-13 or R'

country_ukraine_russia_df = rated_df[np.logical_or(rated_df.country.str.contains('ukraine'),
rated_df.country.str.contains('russia'))]

country_usa_df = rated_df[rated_df.country.str.contains('usa')]

country_others_df = rated_df[~np.logical_or(np.logical_or(rated_df.country.str.contains('ukraine'),
rated_df.country.str.contains('russia')),
rated_df.country.str.contains('usa') )]

country_not_ukr_rus_df = rated_df[~np.logical_or(rated_df.country.str.contains('ukraine'),
rated_df.country.str.contains('russia'))]


# We goes one by one, and find for which group our film belong, than we
# find films from whole dataset which hits into same group
# skip_country False mean we dont filter usa, europe , we take all countries exept ukraine and russia
# include_in_next mean that films  from whole dataset could behave to another group
# not_exactly_tag means that tag have to happens N times
hard_hierarchy = {
'group_GENRE_animation_and_TAG_minion':{
'type':'Soup_just_1',
'soup':['genre_animation', 'minion'],
'skip_country': False,
'include_in_next': True
},
'group_GENRE_animation_and_TAG_racing':{
'type':'Soup_just_1',
'soup':['genre_animation', 'racing'],
'skip_country': False,
'include_in_next': True
},
'group_GENRE_animation_and_TAG_superhero':{
'type':'Soup_just_1',
'soup':['genre_animation', 'superhero'],
'skip_country': False,
'include_in_next': True
},
'group_GENRE_animation_and_TAG_disney':{
'type':'Soup_just_1',
'soup':['genre_animation', 'disney'],
'skip_country': False,
'include_in_next': True
},
'group_GENRE_animation_AND_musical_OR_animation_AND_music':{
'type':'Soup_OR_2x2',
'soup':[['genre_animation', 'genre_musical'],
['genre_animation', 'genre_music']],
'skip_country': False,
'include_in_next': True
},
'group_GENRE_animation_AND_TAG_dragon':{
'type':'Soup_just_1',
'soup':['genre_animation', 'dragon'],
'skip_country': False,
'include_in_next': True
},
'group_GENRE_animation_AND_comedy':{
```

```
'type':'Soup_just_1',
'soup':['genre_animation','genre_comedy'],
'skip_country': False,
'include_in_next': True
},
'group_GENRE_animation':{
'type':'Soup_just_1',
'soup':['genre_animation'],
'skip_country': False,
'include_in_next': False
},

'group_TAG_marvel_comic':{
'type':'Soup_just_1',
'soup':['marvel_comics'],
'skip_country': False,
'include_in_next': True
},
'group_TAG_dc_comic':{
'type':'Soup_just_1',
'soup':['dc_comics'],
'skip_country': False,
'include_in_next': True
},
'group_TAG_based_on_comic_book':{
'type':'Soup_just_1',
'soup':['based_on_comic_book'],
'skip_country': False,
'include_in_next': True
},

'group_GENRE_musical_and_TAG_dancing':{
'type':'Soup_just_1',
'soup':['genre_musical', 'dancing'],
'skip_country': True,
'include_in_next': True
},
'group_GENRE_musical':{
'type':'Soup_just_1',
'soup':['genre_musical'],
'skip_country': False,
'include_in_next': True
},
'group_GENRE_family_AND_TAG_animal':{
'type':'Soup_just_1_notEXACTLY_just_1',
'soup':['genre_family'],
'not_exactly_tag':['animal'],
'times':3,
'skip_country': True,
'include_in_next': True
},

# This must be here! not above not under
'group_TAG_disney':{
'type':'Soup_just_1',
'soup':['disney'],
'skip_country': True,
'include_in_next': True
},

'group_GENRE_drama_AND_family':{
'type':'Soup_just_1',
'soup':['genre_drama', 'genre_family'],
```

```
'skip_country': False,
'include_in_next': True
},

'group_GENRE_music_TAG_dancing':{
'type':'Soup_just_1',
'soup':['genre_music', 'dancing'],
'skip_country': True,
'include_in_next': False
},
'group_GENRE_biography_music_OR_history_music_OR_drama_music':{
'type':'Soup_OR_3x2',
'soup':[['genre_biography', 'genre_music'],
['genre_history', 'genre_music'],
['genre_drama', 'genre_music']],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_music':{
'type':'Soup_just_1',
'soup':['genre_music'],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_documentary_war_OR_history_war_OR_biography_war':{
'type':'Soup_OR_3x2',
'soup':[['genre_documentary', 'genre_war'],
['genre_history', 'genre_war'],
['genre_biography', 'genre_war']],
'skip_country': False,
'include_in_next': False
},

'group_TAG_spy':{
'type':'Soup_just_1',
'soup':['spy'],
'skip_country': False,
'include_in_next': True
},


'group_GENRE_drama_AND_war':{
'type':'Soup_just_1',
'soup':['genre_drama', 'genre_war'],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_documentary_OR_history_OR_biography_AND_TAG_world_war_two':{
'type':'Soup_OR_3x2',
'soup':[['genre_documentary', 'world_war_two'],
['genre_history', 'world_war_two'],
['genre_biography', 'world_war_two']],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_crime_AND_TAG_gangster':{
'type':'Soup_just_1',
'soup':['genre_crime', 'gangster'],
'skip_country': False,
'include_in_next': True
},
'group_GENRE_sport_AND_comedy':{
'type':'Soup_just_1',
```

```
'soup':['genre_sport', 'genre_comedy'],
'skip_country': False,
'include_in_next': True
},
'group_GENRE_sport':{
'type':'Soup_just_1',
'soup':['genre_sport'],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_biography_drama_OR_history_drama_OR_documentary_drama':{
'type':'Soup_OR_3x2',
'soup':[['genre_biography', 'genre_drama'],
['genre_history', 'genre_drama'],
['genre_documentary', 'genre_drama']],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_western':{
'type':'Soup_just_1',
'soup':['genre_western'],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_comedy_AND_horror':{
'type':'Soup_just_1',
'soup':['genre_comedy', 'genre_horror'],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_comedy_AND_TAG_zombie':{
'type':'Soup_just_1',
'soup':['genre_comedy', 'zombie'],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_comedy_AND_TAG_kitchen':{
'type':'Soup_just_1',
'soup':['genre_comedy', 'kitchen'],
'skip_country': False,
'include_in_next': True
},

'group_GENRE_comedy_and_TAGS_love_relationships':{
'type':'genre_tags_or_11',
'soup':[['genre_comedy', 'husband_wife_relationship'],
['genre_comedy', 'boyfriend_girlfriend_relationship'],
['genre_comedy', 'ex_husband_ex_wife_relationship'],
['genre_comedy', 'interracial_relationship'],
['genre_comedy', 'older_man_younger_woman_relationship'],
['genre_comedy', 'ex_boyfriend_ex_girlfriend_relationship'],
['genre_comedy', 'older_woman_younger_man_relationship'],
['genre_comedy', 'male_female_relationship'],
['genre_comedy', 'fiance_fiancee_relationship'],
['genre_comedy', 'ex_girlfriend_ex_boyfriend_relationship'],
['genre_comedy', 'girlfriend_boyfriend_relationship']],
'skip_country': False,
'include_in_next': True
},

'group_GENRE_comedy_and_TAGS_family_relationships':{
'type':'genre_tags_or_21',
'soup':[['genre_comedy', 'father_son_relationship'],
```

```
['genre_comedy', 'father_daughter_relationship'],
['genre_comedy', 'mother_son_relationship'],
['genre_comedy', 'mother_daughter_relationship'],
['genre_comedy', 'brother_sister_relationship'],
['genre_comedy', 'brother_brother_relationship'],
['genre_comedy', 'family_relationships'],
['genre_comedy', 'sister_sister_relationship'],
['genre_comedy', 'aunt_niece_relationship'],
['genre_comedy', 'uncle_nephew_relationship'],
['genre_comedy', 'grandfather_grandson_relationship'],
['genre_comedy', 'cousin_cousin_relationship'],
['genre_comedy', 'grandmother_granddaughter_relationship'],
['genre_comedy', 'aunt_nephew_relationship'],
['genre_comedy', 'uncle_niece_relationship'],
['genre_comedy', 'grandfather_granddaughter_relationship'],
['genre_comedy', 'grandmother_grandson_relationship'],
['genre_comedy', 'stepfather_stepdaughter_relationship'],
['genre_comedy', 'father_in_law_daughter_in_law_relationship'],
['genre_comedy', 'mother_in_law_daughter_in_law_relationship'],
['genre_comedy', 'father_in_law_son_in_law_relationship']],
'skip_country': False,
'include_in_next': True
},

'group_GENRE_comedy_AND_drama':{
'type':'Soup_just_1',
'soup':['genre_comedy', 'genre_drama'],
'skip_country': False,
'include_in_next': True
},
'group_GENRE_comedy_AND_TAG_sex':{
'type':'Soup_just_1_notEXACTLY_just_1',
'soup':['genre_comedy'],
'not_exactly_tag':['sex'],
'times':3,
'skip_country': False,
'include_in_next': False
},

'group_GENRE_comedy_or_adventure_and_tag_car_truck_race_motorcycle_bike':{
'type':'genre_tags_or_10',
'soup':[[['genre_comedy'], ['car']],
[['genre_comedy'], ['truck']],
[['genre_comedy'], ['race']],
[['genre_comedy'], ['motorcycle']],
['genre_comedy', 'bike'],
[['genre_adventure'], ['car']],
[['genre_adventure'], ['truck']],
[['genre_adventure'], ['race']],
[['genre_adventure'], ['motorcycle']],
[['genre_adventure'], ['bike']]],
'skip_country': False,
'include_in_next': True
},


'group_GENRE_comedy':{
'type':'Soup_just_1',
'soup':['genre_comedy'],
'skip_country': False,
'include_in_next': False
},
'group_TAG_zombie':{
```

```
'type':'Soup_just_1',
'soup':['zombie'],
'skip_country': False,
'include_in_next': True
},

'group_GENRE_horror_OR_drama_AND_TAG_post_apocalypse': {
'type': 'Soup_OR_2x2',
'soup': [['genre_horror', 'post_apocalypse'],
['genre_drama', 'post_apocalypse']],
'skip_country': False,
'include_in_next': True
},

'group_GENRE_horror_AND_TAG_time_loop':{
'type':'Soup_just_1',
'soup':['genre_horror', 'time_loop'],
'skip_country': False,
'include_in_next': True
},


'group_GENRE_horror_AND_mystery_AND_thriller':{
'type':'Soup_just_1',
'soup':['genre_horror', 'genre_mystery', 'genre_thriller'],
'skip_country': False,
'include_in_next': True
},

'group_GENRE_drama_OR_thriller_AND_TAG_superhero':{
'type':'Soup_OR_2x2',
'soup':[['genre_thriller', 'superhero'],
['genre_drama', 'superhero']],
'skip_country': False,
'include_in_next': True
},

'group_GENRE_scifi_AND_TAG_female_protagonist':{
'type':'Soup_just_1',
'soup':['genre_sci-fi', 'female_protagonist'],
'skip_country': False,
'include_in_next': True
},

'group_GENRE_action_AND_TAG_female_protagonist':{
'type':'Soup_just_1',
'soup':['genre_action', 'female_protagonist'],
'skip_country': False,
'include_in_next': True
},

'group_GENRE_horror_AND_TAG_shark':{
'type':'Soup_just_1_notEXACTLY_just_1',
'soup':['genre_horror'],
'not_exactly_tag':['shark'],
'times':3,
'skip_country': True,
'include_in_next': True
},

'group_GENRE_horror_AND_TAGS_sea_shark_ocean_animal':{
'type':'genre_tags_or_8',
'soup':[['genre_horror'], ['sea']],
```

```
[['genre_horror'], ['shark']],
[['genre_horror'], ['ocean']],
[['genre_horror'], ['dog']],
[['genre_horror'], ['cat']],
[['genre_horror'], ['tiger']],
[['genre_horror'], ['lion']],
[['genre_horror'], ['spider']]],
'skip_country': True,
'include_in_next': True
},


'group_GENRE_scifi_AND_horror':{
'type':'Soup_just_1',
'soup':['genre_sci-fi', 'genre_horror'],
'skip_country': False,
'include_in_next': True
},
'group_GENRE_sci-fi_AND_TAG_space_travel_OR_space':{
'type':'Soup_OR_2x2',
'soup':[['genre_sci-fi', 'space_travel'],
['genre_sci-fi', 'space']],
'skip_country': False,
'include_in_next': True
},
'group_GENRE_sci-fi_AND_TAG_martial_arts':{
'type':'Soup_just_1',
'soup':['genre_sci-fi', 'martial_arts'],
'skip_country': False,
'include_in_next': True
},
'group_GENRE_sci-fi_AND_TAG_survivor':{
'type':'Soup_just_1',
'soup':['genre_sci-fi', 'survivor'],
'skip_country': False,
'include_in_next': True
},
'group_GENRE_thriller_AND_horror':{
'type':'Soup_just_1',
'soup':['genre_thriller', 'genre_horror'],
'skip_country': False,
'include_in_next': True
},
'group_GENRE_action_AND_thriller':{
'type':'Soup_just_1',
'soup':['genre_action', 'genre_thriller'],
'skip_country': False,
'include_in_next': True
},
'group_GENRE_adventure_AND_family_AND_fantasy':{
'type':'Soup_just_1',
'soup':['genre_adventure', 'genre_family', 'genre_fantasy'],
'skip_country': False,
'include_in_next': True
},


'group_GENRE_scifi':{
'type':'Soup_just_1',
'soup':['genre_sci-fi'],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_biography':{
```

```
'type':'Soup_just_1',
'soup':['genre_biography'],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_history':{
'type':'Soup_just_1',
'soup':['genre_history'],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_documentary':{
'type':'Soup_just_1',
'soup':['genre_documentary'],
'skip_country': False,
'include_in_next': False
},
'group_TAG_alien':{
'type':'Soup_just_1',
'soup':['alien'],
'skip_country': False,
'include_in_next': True
},
'group_GENRE_drama_and_TAGS_love_relationships':{
'type':'genre_tags_or_11',
'soup':[['genre_drama', 'husband_wife_relationship'],
['genre_drama', 'boyfriend_girlfriend_relationship'],
['genre_drama', 'ex_husband_ex_wife_relationship'],
['genre_drama', 'interracial_relationship'],
['genre_drama', 'older_man_younger_woman_relationship'],
['genre_drama', 'ex_boyfriend_ex_girlfriend_relationship'],
['genre_drama', 'older_woman_younger_man_relationship'],
['genre_drama', 'male_female_relationship'],
['genre_drama', 'fiance_fiancee_relationship'],
['genre_drama', 'ex_girlfriend_ex_boyfriend_relationship'],
['genre_drama', 'girlfriend_boyfriend_relationship']],
'skip_country': False,
'include_in_next': True
},

'group_GENRE_drama_and_TAGS_family_relationships':{
'type':'genre_tags_or_21',
'soup':[['genre_drama', 'father_son_relationship'],
['genre_drama', 'father_daughter_relationship'],
['genre_drama', 'mother_son_relationship'],
['genre_drama', 'mother_daughter_relationship'],
['genre_drama', 'brother_sister_relationship'],
['genre_drama', 'brother_brother_relationship'],
['genre_drama', 'family_relationships'],
['genre_drama', 'sister_sister_relationship'],
['genre_drama', 'aunt_niece_relationship'],
['genre_drama', 'uncle_nephew_relationship'],
['genre_drama', 'grandfather_grandson_relationship'],
['genre_drama', 'cousin_cousin_relationship'],
['genre_drama', 'grandmother_granddaughter_relationship'],
['genre_drama', 'aunt_nephew_relationship'],
['genre_drama', 'uncle_niece_relationship'],
['genre_drama', 'grandfather_granddaughter_relationship'],
['genre_drama', 'grandmother_grandson_relationship'],
['genre_drama', 'stepfather_stepdaughter_relationship'],
['genre_drama', 'father_in_law_daughter_in_law_relationship'],
['genre_drama', 'mother_in_law_daughter_in_law_relationship'],
['genre_drama', 'father_in_law_son_in_law_relationship']],
```

```
'skip_country': False,
'include_in_next': True
},

'group_GENRE_drama_AND_romance':{
'type':'Soup_just_1',
'soup':['genre_drama', 'genre_romance'],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_drama_AND_thriller':{
'type':'Soup_just_1',
'soup':['genre_drama', 'genre_thriller'],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_fantasy_AND_adventure':{
'type':'Soup_just_1',
'soup':['genre_fantasy', 'genre_adventure'],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_fantasy':{
'type':'Soup_just_1',
'soup':['genre_fantasy'],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_action':{
'type':'Soup_just_1',
'soup':['genre_action'],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_adventure':{
'type':'Soup_just_1',
'soup':['genre_adventure'],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_horror':{
'type':'Soup_just_1',
'soup':['genre_horror'],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_thriller':{
'type':'Soup_just_1',
'soup':['genre_thriller'],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_drama':{
'type':'Soup_just_1',
'soup':['genre_drama'],
'skip_country': False,
'include_in_next': False
},
'group_GENRE_romance':{
'type':'Soup_just_1',
'soup':['genre_romance'],
'skip_country': False,
'include_in_next': False
```

```
    },
    'group_GENRE_family':{
        'type':'Soup_just_1',
        'soup':['genre_family'],
        'skip_country': False,
        'include_in_next': False
    },
    'group_GENRE_crime':{
        'type':'Soup_just_1',
        'soup':['genre_crime'],
        'skip_country': False,
        'include_in_next': False
    },
    'group_GENRE_mystery':{
        'type':'Soup_just_1',
        'soup':['genre_mystery'],
        'skip_country': False,
        'include_in_next': False
    },
    'group_GENRE_war':{
        'type':'Soup_just_1',
        'soup':['genre_war'],
        'skip_country': False,
        'include_in_next': False
    },
    'group_GENRE_short':{
        'type':'Soup_just_1',
        'soup':['genre_short'],
        'skip_country': False,
        'include_in_next': False
    }

}
def main(model_name='default', save_model=True, debug = True, LONG_ID='tt5884230'):

    # Get data about films
    #df = SQL_BASE_user()
    #df.to_csv('SQL_BASE_user.csv', index=False)
    df = pd.read_csv('SQL_BASE_user.csv', dtype={'long':str}).fillna('')

    # make pairs for train nn
    pairs, pairs_set, tag_index, film_index, index_film, tags, films = make_pairs_for_train_NN(df)


    if save_model:
        model = train_model(pairs, pairs_set, tag_index, film_index, tags, films, epoch=epoch_amount,
                            batch_size=256)
        model.save(model_name + '.h5')
    else:
        model = load_model(model_name + '.h5')

    # debug one film
    if debug:
        # find percentage similarity to each film from dataset
        closest_films, distances = get_similar_film_by_nn(df[df.long == LONG_ID].long.values[0], model,
                                                          film_index, index_film)
        distances = np.interp(distances, (distances.min(), distances.max()), (0, +0.98))
        zip_film_score = list(zip(closest_films, distances))

        # if kinomama or origVoice we search from kinomamas data set first, than from all dataset
        if df[df.long == LONG_ID]['status'].values[0] == 'КіноМама':
            df_mama = df[df.status == 'КіноМама']
            chunk_mama = get_chunk(df_mama, df[df.long == LONG_ID].long.values[0], closest_films, film_index)
```

```
# If becouse of some films couldnt contain any information
if chunk_mama != 'WRONG_FILM_PARAMETER':
chunk_mama = [[i[0], 'Mama ' + i[1]] for i in chunk_mama]
else:
chunk_mama = None

chunk_mama = [[i[0], 'Mama ' + i[1]] for i in chunk_mama]
df_additional = df[np.logical_and(np.logical_and(df.status != 'In English', df.status != 'КіноМама'),
df.imdbID != df[df.long == LONG_ID].imdbID.values[0])]
df_additional = df_additional.append(df[df.long == LONG_ID])
chunk = get_chunk(df_additional, df[df.long == LONG_ID].long.values[0], closest_films, film_index,
chunk_mama)

elif df[df.long == LONG_ID]['status'].values[0] == 'In English':
df_orig_rec = df[df.status == 'In English']
chunk_orig = get_chunk(df_orig_rec, df[df.long == LONG_ID].long.values[0], closest_films, film_index)
# If becouse of some films couldnt contain any information
if chunk_orig != 'WRONG_FILM_PARAMETER':
chunk_orig = [[i[0], 'InEnglish ' + i[1]] for i in chunk_orig]
else:
chunk_orig = None
df_additional = df[np.logical_and(np.logical_and(df.status != 'In English', df.status != 'КіноМама'),
df.imdbID != df[df.long == LONG_ID].imdbID.values[0])]
df_additional = df_additional.append(df[df.long == LONG_ID])
chunk = get_chunk(df_additional, df[df.long == LONG_ID].long.values[0], closest_films, film_index,
chunk_orig)

else:
df_without_mama_and_english = df[np.logical_and(df.status != 'КіноМама', df.status != 'In English')]
chunk = get_chunk(df_without_mama_and_english, df[df.long == LONG_ID].long.values[0], closest_films,
film_index)


if chunk != 'WRONG_FILM_PARAMETER':
chunk = pd.DataFrame(chunk, columns=['indexx', 'Category'])
zip_film_score = pd.DataFrame(zip_film_score, columns=['indexx', 'Coef'])
result = pd.merge(chunk, zip_film_score, how='inner', on=['indexx'])
result['indexx'] = [index_film[i] for i in result['indexx'].tolist()]
res_title = [df[df.long == i].originalTitle.values[0] for i in result['indexx'].tolist()]
result['res_title'] = res_title


# make percentage for additions lover
result.loc[result[result.Category.str.contains('ADD_#1')]['Coef'].index, 'Coef'] =
result[result.Category.str.contains('ADD_#1')]['Coef'] / 7
result.loc[result[result.Category.str.contains('ADD_#2')]['Coef'].index, 'Coef'] =
result[result.Category.str.contains('ADD_#2')]['Coef'] / 14
result.loc[result[result.Category.str.contains('director')]['Coef'].index, 'Coef'] = 0.99

result['Coef'] = result['Coef'].round(2) * 100
result['Coef'] = result['Coef'].astype(int)
print(result[['res_title', 'Category', 'Coef']])
else:
print('WRONG_FILM_PARAMETER')

else:

result = pd.DataFrame()

for index, row in tqdm(df.iterrows()):
# find percentage similarity to each film from dataset
closest_films, distances = get_similar_film_by_nn(row['long'], model, film_index, index_film)
```

```
zip_film_score = list(zip(closest_films, distances))

# if kinomama or origVoice we search from kinomamas data set first, than from all dataset
if row['status'] == 'КіноМама':
df_mama = df[df.status == 'КіноМама']
chunk_mama = get_chunk(df_mama, row['long'], closest_films, film_index)
mama_imdbs = df_mama.imdbID.values

# If becouse of some films couldnt contain any information
if chunk_mama != 'WRONG_FILM_PARAMETER':
chunk_mama = [[i[0], 'Mama ' + i[1]] for i in chunk_mama]
else:
chunk_mama = None
df_additional = df[np.logical_and(np.logical_and(np.logical_and(df.status != 'In English', df.status !=
'КіноМама'), df.imdbID != row['imdbID']), ~df.imdbID.isin(mama_imdbs))]
df_additional = df_additional.append(row)
chunk = get_chunk(df_additional, row['long'], closest_films, film_index, chunk_mama)

elif row['status'] == 'In English':
df_orig_rec = df[df.status == 'In English']
chunk_orig = get_chunk(df_orig_rec, row['long'], closest_films, film_index)
df_orig_rec_imdbs = df_orig_rec.imdbID.values

# If becouse of some films couldnt contain any information
if chunk_orig != 'WRONG_FILM_PARAMETER':
chunk_orig = [[i[0], 'InEnglish ' + i[1]] for i in chunk_orig]
else:
chunk_orig = None
df_additional = df[np.logical_and(np.logical_and(np.logical_and(df.status != 'In English', df.status !=
'КіноМама'), df.imdbID != row['imdbID']), ~df.imdbID.isin(df_orig_rec_imdbs))]
df_additional = df_additional.append(row)
chunk = get_chunk(df_additional, row['long'], closest_films, film_index, chunk_orig)

else:
df_without_mama_and_english = df[np.logical_and(df.status != 'КіноМама', df.status != 'In English')]
chunk = get_chunk(df_without_mama_and_english, row['long'], closest_films, film_index)


if chunk != 'WRONG_FILM_PARAMETER':

chunk = pd.DataFrame(chunk, columns=['indexx', 'Category'])
zip_film_score = pd.DataFrame(zip_film_score, columns=['indexx', 'Coef'])
temp = pd.merge(chunk, zip_film_score, how='inner', on=['indexx'])
temp['indexx'] = [index_film[i] for i in temp['indexx'].tolist()]
lon_rec = [df[df.long == i].long.values[0] for i in temp['indexx'].tolist()]
temp['long_orig'] = df[df.long == row['long']].long.values[0]
temp['long_rec'] = lon_rec


temp.drop_duplicates(subset=['indexx'], inplace=True)
temp = temp[:films_to_show]


add_2_coefs = temp[temp.Category.str.contains('ADD_#2')]['Coef'].values
add_1_coefs = temp[temp.Category.str.contains('ADD_#1')]['Coef'].values
main_coefs = temp[~temp.Category.str.contains('ADD_#')]['Coef'].values


# scale coefitients
try:
temp.loc[temp[~temp.Category.str.contains('ADD_#')]['Coef'].index, 'Coef'] = np.interp(main_coefs,
(main_coefs.min(), main_coefs.max()), (0.6, 0.97))
```

```
                except:
                    print('warning! while coef was tried change it was empty (when scale coefs)')
                try:
                    temp.loc[temp[temp.Category.str.contains('ADD_#1')]['Coef'].index, 'Coef'] = np.interp(add_1_coefs,
                                                                                                           (add_1_coefs.min(), add_1_coefs.max()), (0.3, 0.59))
                except:
                    print('warning! while coef was tried change it was empty (when scale coefs)')
                try:
                    temp.loc[temp[temp.Category.str.contains('ADD_#2')]['Coef'].index, 'Coef'] = np.interp(add_2_coefs,
                                                                                                           (add_2_coefs.min(), add_2_coefs.max()), (0.02, 0.29))
                except:
                    print('warning! while coef was tried change it was empty (when scale coefs)')

                temp.loc[temp[temp.Category.str.contains('sequel_prequel')]['Coef'].index, 'Coef'] = 0.98
                temp.loc[temp[temp.Category.str.contains('director')]['Coef'].index, 'Coef'] = 0.98
                temp.loc[temp[temp.Category.str.contains('Mama')]['Coef'].index, 'Coef'] = 0.99
                temp.loc[temp[temp.Category.str.contains('InEnglish')]['Coef'].index, 'Coef'] = 0.99

                temp['Coef'] = temp['Coef'].round(2) * 100
                temp['Coef'] = temp['Coef'].astype(int)


                result = result.append(temp[['long_orig', 'long_rec', 'Coef', 'Category']])



            else:
                print(row)
                print(row['long'])
                print('WRONG_FILM_PARAMETER (film does not contain any genre and keywords)')


        result.to_csv('result.csv', index=False)


    main(model_name='epoch_201', save_model = True, debug = True,
         LONG_ID='00000000000000000000000000000910') import pandas as pd
    from scipy.spatial.distance import euclidean
    import requests
    import scipy
    import warnings
    import datetime
from loadings import client_movie_SQL, SQL_BASE_user, favorites_films, load_recommendations, dwh_all_clients, \
    SurveyMonkey, movie_cinema, last_cinema, dwh_client_agelimit, dwh_film_agelimit, \
    films_which_will_ordered, film_genre_description

    import tqdm

    from paralelize import parallelize_dataframe

    warnings.filterwarnings("ignore")
    tqdm.tqdm.pandas()


    # How many films take from recomendations system
    n_films_from_recomendation_system = 20
    n_films_to_recommend = 10

    print('Part 1 started', datetime.datetime.now())
    start_time = datetime.datetime.now()

    try:
```

```
print('Start Get data (client|film which he has alredy watched) from MSSQL')
# Get data -> (client|film which he has alredy watched)
client_film = client_movie_SQL()
print('End load from MSSQL')
client_film.ID = client_film.ID.apply(lambda x: x.strip())
client_film.rename(columns={'movie': 'MuvieId-suggested'}, inplace=True)

print('Start Get data from MySql:')
print('# 1) imdb info for each film')
print('# 2) movie list, for grep data which now in cinema now')
df, df_movie_list = SQL_BASE_user()

print('End load from MySql')

# get films which now in cinema
films_for_oredring = films_which_will_ordered()
now_in_cinema = df_movie_list[df_movie_list['movieUid'].isin(films_for_oredring.longID.values)]
now_in_cinema.rename(columns={'movieUid':'MovieId-2'},inplace=True)

# Add liked and not watched films to alredy watched films
# We will assign 1 and -1 later
client_film['is_like'] = 0

# ================================  Add Favorites
================================

# FAVORITES HERE BECOUSE WE THINK THAT LIKED FILMS ALREDY WATCHED!!!!!!
favorites = favorites_films(now_in_cinema['MovieId-2'].values)
client_film = pd.concat([client_film, favorites.rename(columns={'film':'MuvieId-suggested'})])
# keep 'first' becouse of if client have alredy watched
# there is will be 0 in 'is_like' field
client_film = client_film.drop_duplicates(['ID','MuvieId-suggested'])


# recommendations for films which is showing now in cinema
recomend_in_prokat = load_recommendations(now_in_cinema['MovieId-2'].values)
recomend_in_prokat = recomend_in_prokat.sort_values(['MovieId-chosen', 'Percent'])

# get last {n_films_from_recomendation_system} records (much more relatve) in group (becouse of ascending
sort)
recomend_in_prokat = recomend_in_prokat.groupby('MovieId-
chosen').tail(n_films_from_recomendation_system)

# Merge for make reccomendation for clients
tmp = pd.merge(client_film, recomend_in_prokat, how='inner', on=['MuvieId-suggested'])

# Merge data (if client has watched simillar film to film which now in cinema, than we reccomend this film to him,
but
# we must count mean bocouse of he could watch two cimilar films)
reccomendation_content_base = tmp.groupby(['ID', 'MovieId-
chosen']).mean()['Percent'].reset_index().sort_values(
['ID', 'Percent'])

# scale walues fron 1 to 0.5 bocouse we supposed that content base system gives better results than part 2 (see
next)
reccomendation_content_base['scaled_values'] =
reccomendation_content_base.groupby('ID')['Percent'].transform(
lambda x: 0.48 * (x - min(x)) / (max(x) - min(x)) + 0.51)

# fillna 1 becouse if client has 1 film than max(x) - min(x) = 0
reccomendation_content_base.scaled_values[reccomendation_content_base.scaled_values.isna()] =
reccomendation_content_base.Percent[reccomendation_content_base.scaled_values.isna()] / 100
```

```
print('stage 1 done')

# ================================ Second Part ================================

# Get data (description for make client pattern)
film_description = film_genre_description(df)
film_description.rename(columns={'long': 'MuvieId-suggested'}, inplace=True)

# As result we will have client|list of ganres which he love from 0 to 1
user_preferences = pd.merge(client_film.drop(columns=['is_like']), film_description, how='inner',
                            on=['MuvieId-suggested'])
user_genre_preferences = user_preferences.groupby('ID').mean()
user_genre_preferences = user_genre_preferences.reset_index()

# Describe film by genres
film_description_now_in_cinema = film_description[
film_description.apply(lambda x: x['MuvieId-suggested'] in now_in_cinema['MovieId-2'].values, axis=1)]

# Find distance between user and film which now in cinema
rec_2 = scipy.spatial.distance.cdist(user_genre_preferences.set_index('ID'),
                                     film_description_now_in_cinema.set_index('MuvieId-suggested'), metric='cosine')


# make data frame with reccomendation
reccomender_2 = pd.DataFrame(rec_2)
reccomender_2.index = user_genre_preferences.set_index('ID').index
reccomender_2.columns = film_description_now_in_cinema.set_index('MuvieId-suggested').index

# Transform from wide to long format
reccomender_2_flipped = pd.DataFrame()
for col_number in range(len(reccomender_2.columns)):
a = pd.DataFrame(reccomender_2.iloc[:, col_number]).rename(
columns={pd.DataFrame(reccomender_2.iloc[:, col_number]).columns[0]: 'value'})
a['film'] = pd.DataFrame(reccomender_2.iloc[:, col_number]).columns[0]
reccomender_2_flipped = reccomender_2_flipped.append(pd.DataFrame(a))

reccomender_2_flipped = reccomender_2_flipped.reset_index()
reccomender_2_flipped = reccomender_2_flipped.sort_values(['ID', 'film'])

# scale from 0 to 0.5 becouse we suppose that thist type of reccomendation is weaker than content-based
approach
reccomender_2_flipped['scaled_values'] = reccomender_2_flipped.groupby('ID')['value'].transform(
lambda x: 0.47 * (x - min(x)) / (max(x) - min(x)) + 0.02)

reccomender_2_flipped.scaled_values = 1 - reccomender_2_flipped.scaled_values
reccomender_2_flipped['scaled_values'] = reccomender_2_flipped.groupby('ID')['scaled_values'].transform(
lambda x: 0.47 * (x - min(x)) / (max(x) - min(x)) + 0.02)

print('stage 2 done')


# ================================ Merge ================================


# Merge 2 types of reccomendations
reccomendation_content_base.rename(columns={'MovieId-chosen': 'film'}, inplace=True)
reccomendation_content_base = reccomendation_content_base[['ID', 'film', 'scaled_values']]
reccomendation_content_base = reccomendation_content_base.append(reccomender_2_flipped[['ID', 'film',
'scaled_values']])
reccomendation_content_base = reccomendation_content_base.groupby(['ID', 'film']).max().reset_index()
```

```
reccomendation_content_base.scaled_values = reccomendation_content_base.scaled_values.apply(lambda x:
round(x, 2))


# ================================ recomendation for user who doesn watch any movie
================================

dwh_all_cl = dwh_all_clients()
dwh_all_cl.ID = dwh_all_cl.ID.apply(lambda x: x.strip())

clients_who_saw_at_least_one_movie = set(reccomendation_content_base.ID.unique())
clients_all = set(dwh_all_cl.ID.unique())
clients_who_didnt_saw_anything = clients_all.difference(clients_who_saw_at_least_one_movie)

clients_who_didnt_saw_anything_df = pd.DataFrame(list(clients_who_didnt_saw_anything), columns=['ID'])


top_favorites_film = favorites.groupby('film').sum().reset_index().sort_values(['is_like'], ascending=False)
top_favorites_film['scaled_values'] = top_favorites_film['is_like'].transform(
lambda x: 0.9756248 * (x - min(x)) / (max(x) - min(x)) + 0.00894664)

top_favorites_film.scaled_values = top_favorites_film.scaled_values.apply(lambda x: round(x, 2))

for index, row in tqdm.tqdm(top_favorites_film.iterrows()):
clients_who_didnt_saw_anything_df['film'] = row['film']
clients_who_didnt_saw_anything_df['scaled_values'] = row['scaled_values']

reccomendation_content_base = reccomendation_content_base.append(clients_who_didnt_saw_anything_df)


recomender_with_favorites = pd.merge(reccomendation_content_base, favorites, how='left', on=['ID',
'film']).fillna(0)
recomender_with_favorites = recomender_with_favorites.sort_values(['ID', 'is_like', 'scaled_values'],
ascending=False)

# assing 1 to films which in favorites
mask = recomender_with_favorites.is_like == 1
recomender_with_favorites.loc[mask, 'scaled_values'] = 1


# ================================ Survey Monkey
================================


survey_monkey = SurveyMonkey(client_film['MuvieId-suggested'].unique())
# Delete duplicates becouse problems in db (duplicates are present)
survey_monkey = survey_monkey.dropna()

# If film any film which similar to film which now in cinema is disliked, we assign to this film 0
recomend_to_survey = load_recommendations(list(survey_monkey['film'].unique()))
recomend_to_survey = recomend_in_prokat[recomend_in_prokat['MuvieId-
suggested'].isin(now_in_cinema['MovieId-2'].values)]

recomend_to_survey['survey'] = 0
recomend_to_survey.rename(columns={'MovieId-chosen': 'film'}, inplace=True)
survey_delete = pd.merge(survey_monkey, recomend_to_survey, how='inner', on=['film'])
survey_delete = survey_delete[['ID', 'MuvieId-suggested', 'survey']]
survey_delete.rename(columns={'MuvieId-suggested': 'film'}, inplace=True)
```

```
recomender_with_favorites_survey = pd.merge(recomender_with_favorites, survey_delete, how='left',
                                            on=['ID', 'film']).fillna(-1)


# assing 0 to films which in survey_monkey
mask = recomender_with_favorites_survey.survey == 0
recomender_with_favorites_survey.loc[mask, 'scaled_values'] = 0

# ================================  Delete alredy watched films from recomendations
================================


# Delete films which alredy watched by client
recomender_with_favorites_survey = recomender_with_favorites_survey[['ID', 'film', 'scaled_values']]

client_film_alredy_watched = client_film[(client_film['MuvieId-suggested'].isin(now_in_cinema['MovieId-
2'].values)) & (client_film.is_like == 0)]
client_film_alredy_watched.rename(columns={'MuvieId-suggested':'film'}, inplace=True)
client_film_alredy_watched['to_delete'] = 1


result_recomendation = pd.merge(recomender_with_favorites_survey, client_film_alredy_watched, how='left',
                                on=['ID', 'film'])
result_recomendation = result_recomendation[result_recomendation.to_delete.isna()]

result_recomendation = result_recomendation[['ID', 'film', 'scaled_values']]


result_recomendation.scaled_values = result_recomendation.scaled_values * 100
result_recomendation['scaled_values'] = result_recomendation['scaled_values'].round(2)

print('stage 3 done')

# ================================  Recomendation by Last Town
================================


#add city
#Movies in cinema ( in different cinemas shows different films)
dict_movie_in_each_cinema=movie_cinema()
client_last_cinema=last_cinema()
client_last_cinema.ID = client_last_cinema.ID.apply(lambda x: x.strip())
client_last_cinema.loc[client_last_cinema.Cinema == '2445', 'Cinema'] = '908'


result_recomendation_last_cinema = pd.merge(result_recomendation,client_last_cinema, how='left', on='ID')
result_recomendation_last_cinema.Cinema.fillna('908 ', inplace=True)
result_recomendation_last_cinema['Cinema'] = result_recomendation_last_cinema['Cinema'].apply(lambda x:
dict_movie_in_each_cinema.get(x, dict_movie_in_each_cinema.get('908 ')))

# ===== paralelize =====
# result_recomendation_last_cinema = result_recomendation_last_cinema[
#     result_recomendation_last_cinema[['film', 'Cinema']].progress_apply(lambda x: x.film in x.Cinema, axis=1)]
def func1(df):
    return df[df[['film', 'Cinema']].apply(lambda x: x.film in x.Cinema, axis=1)]
result_recomendation_last_cinema = parallelize_dataframe(result_recomendation_last_cinema, func1)
# ===== paralelize =====

result_recomendation = result_recomendation_last_cinema[['ID', 'film', 'scaled_values']]


# ================================  Filter by client_AGE
================================
```

```
dwh_client_age = dwh_client_agelimit()
dwh_client_age.ID = dwh_client_age.ID.apply(lambda x: x.strip())
dwh_movie_age = dwh_film_agelimit()

result_recomendation = pd.merge(result_recomendation, dwh_client_age, how='left', on=['ID'])

result_recomendation = pd.merge(result_recomendation, dwh_movie_age, how='left', on=['film'])

result_recomendation.Rate_x.fillna('Від 13 до 16', inplace=True)
result_recomendation.Rate_y.fillna('Від 13 до 16', inplace=True)


# ===== paralelize =====
# result_recomendation['age_balance'] = result_recomendation.apply(lambda x: x.Rate_x == x.Rate_y,
axis=1).astype(int)
def func2(df):
df['age_balance'] = df.apply(lambda x: x.Rate_x == x.Rate_y, axis=1).astype(int)
return df
result_recomendation = parallelize_dataframe(result_recomendation, func2)
# ===== paralelize =====

result_recomendation['first_part'] = -1
result_recomendation.loc[result_recomendation.scaled_values >= 51 , 'first_part'] = 1

result_recomendation.loc[result_recomendation.scaled_values == 100, 'age_balance'] = 9999

result_recomendation = result_recomendation.sort_values(['ID', 'first_part', 'scaled_values'], ascending=[True,
False, False])

result_recomendation = result_recomendation[~((result_recomendation.Rate_x != '17 і більше') &
(result_recomendation.Rate_y == '17 і більше') & (result_recomendation.scaled_values != 100))]

result_recomendation = result_recomendation[['ID', 'film', 'scaled_values']]


print('stage 4 done')

result_recomendation.to_csv(r'hard_drive/intermediate_result.csv', index=False)
```